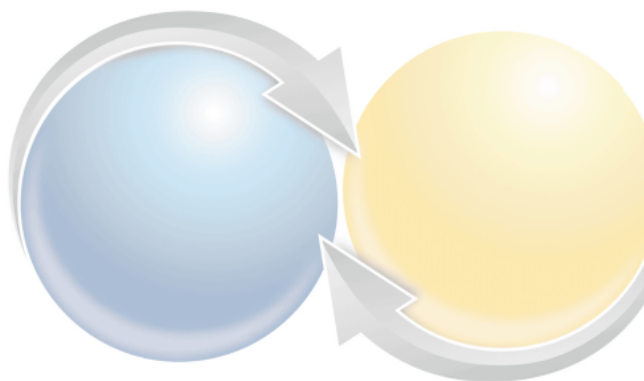


Livelink WCM Server

Portal Manager API Programmierhandbuch

Dieses Handbuch beschreibt die Entwicklung dynamischer und personalisierter Websites auf Grundlage des Portal Manager API. Sie erhalten Informationen zu folgenden Themen:

- grundlegende Konzepte des Portal Manager API
- Konfiguration des Portal Manager API im Admin-Client
- Überblick über die Klassen und Interfaces des Portal Manager API
- Anwendungsbeispiele für die Programmierung mit dem Portal Manager API



Copyright 2005 Open Text Corporation

Das Copyright an diesen Unterlagen und der dazugehörigen Software gehört ohne Einschränkungen Open Text. Diese Unterlagen und die dazugehörige Software dürfen ohne die ausdrückliche, schriftliche Genehmigung von Open Text weder ganz noch teilweise kopiert werden.

Die Open Text Corporation ist Eigentümer der Warenzeichen Open Text, 'Great Minds Working Together', Livelink, MeetingZone u.a.; diese Liste ist nicht vollständig. Andere erwähnte Warenzeichen sind Eigentum des jeweiligen Unternehmens und werden nur zum Zweck der Identifizierung der Produkte und Unternehmen verwendet. Alle Rechte vorbehalten.

Für die in diesem Dokument beschriebene Software der Open Text Corporation gelten bestimmte Gewährleistungen und Einschränkungen. Informationen zu diesen Gewährleistungen und Einschränkungen erhalten Sie in dem Lizenzvertrag, der zwischen dem Lizenznehmer und der Open Text Corporation geschlossen wurde.

Kontaktadresse:

Unternehmenshauptsitz
Open Text Corporation
185 Columbia Street West,
Waterloo, Ontario
N2L 5Z5
Kanada

Telefon: +1 519 888-7111

Wenn Sie Abonnent des Software Maintenance Program sind oder weitere Informationen zu anderen Kundendienstprogrammen wünschen, wenden Sie sich an den Kundendienst von Open Text unter <http://www.opentext.com/services/support.html>.

Wenn Sie zu dieser Veröffentlichung Vorschläge machen möchten, senden Sie eine E-Mail-Nachricht an documentation@opentext.com.

Weitere Informationen zu den Produkten und Dienstleistungen von Open Text finden Sie auf unserer Homepage unter <http://www.opentext.com>.

© 2005 IXOS SOFTWARE AG

Bretonischer Ring 12
85630 Grasbrunn, Deutschland

Tel.: +49 (89) 4629-0

Fax: +49 (89) 4629-1199

E-Mail: [<office@ixos.de>](mailto:office@ixos.de)

Internet: <http://www.ixos.de>

Alle Rechte vorbehalten. Einschließlich solche, die die Reproduktion, das Kopieren oder eine andere Verwendung oder Übermittlung der Inhalte dieses Dokumentes oder Teile davon betreffen. Kein Teil dieser Publikation darf, egal in welcher Form, ohne die schriftliche Zustimmung der IXOS SOFTWARE AG reproduziert, an Dritte übermittelt, unter Einsatz elektronischer Retrieval-Systeme verarbeitet, kopiert, verteilt oder für öffentliche

Vorfürhungen verwendet werden. IXOS behält sich das Recht vor, Aktualisierungen und Änderungen der Inhalte vorzunehmen. Sämtliche Daten, die auf Bildschirmphotographien (screenshots) sichtbar sind, dienen lediglich als Beispiel zur Demonstration der Software. Für den Inhalt dieser Daten übernimmt IXOS keine Gewähr. Dieses Produkt beinhaltet Software, die im Rahmen des Projekts OpenSSL für den Gebrauch im OpenSSL Toolkit (<http://www.openssl.org/>) bzw. die durch die Apache Software Foundation (<http://www.apache.org/>) entwickelt wurde.

Marken IXOS: IXOS SOFTWARE AG.

SAP®, R/3® und SAP ArchiveLink® sind eingetragene Marken der SAP AG.

Microsoft®, Microsoft Windows NT® und die Namen weiterer Microsoft-Produkte sind eingetragene Marken der Microsoft Corporation.

Acrobat Reader Copyright © 1987 Adobe Systems Incorporated. Alle Rechte vorbehalten. Adobe und Acrobat sind Marken von Adobe Systems Incorporated, die in bestimmten Rechtsbereichen registriert sein können.

Siebel® ist eingetragene Marke der Siebel Systems, Inc.

Sonstige Produktnamen werden nur zur Identifikation der Produkte verwendet und können eingetragene Marken der entsprechenden Hersteller sein.

Copyright © 2005 Gauss Interprise AG Hamburg, Gauss Interprise, Inc. Irvine, California. Alle Rechte weltweit vorbehalten.

Dieses Dokument sowie die zugehörige Software sind Eigentum der Gauss Interprise AG oder ihrer Zulieferer und durch Gesetze zum Schutze des Urheberrechts und andere Gesetze geschützt. Sie werden unter einer Lizenz vertrieben, durch welche die Nutzung, Reproduktion, Vertrieb und Dekompilierung eingeschränkt wird. Weder der Erhalt noch der Besitz dieses Dokumentes ermächtigt Sie, dessen Inhalte ganz oder teilweise auf Papier, elektronisch oder einem anderen Medium zu reproduzieren, weiterzugeben oder anderen den Zugang darauf zu ermöglichen. Kein Teil dieses Dokumentes darf in irgendeiner Form und Weise ohne vorherige schriftliche Zustimmung der Gauss Interprise AG oder Gauss Interprise, Inc. reproduziert werden.

Darüber hinaus gelten für diese Dokumentation die Bestimmungen des Softwarelizenzvertrags.

Alle Warenzeichen oder Handelsmarken, die in diesem Dokument erwähnt wurden, sind Eigentum der entsprechenden Firmen.

<http://www.gaussvip.com>

Programmversion: Livelink Web Content Management Server™ (Content Server) 9.5.0

Dokumentenversion: De-01

Erscheinungsdatum: Januar 2005

Inhaltsverzeichnis

Abbildungsverzeichnis		6
Tabellenverzeichnis		8
Kapitel 1	Einleitung	11
	1.1 Hinweise zu diesem Handbuch	12
	1.2 Typographische Konventionen	16
Kapitel 2	Konzepte	19
	2.1 Architektur des Portal Manager API	19
	2.2 Integration des Portal Manager API in das WCM-System	24
	2.3 Programmierkonzepte	33
	2.4 Integration externer Anwendungen	48
Kapitel 3	Portal Manager API verwalten	53
	3.1 Repositories verwalten	55
	3.2 Applications verwalten	69
	3.3 LDAP	91
	3.4 Fehlersuche	94
Kapitel 4	Klassen und Interfaces des Portal Manager API	99
	4.1 Klassendiagramme und Klassen-beschreibungen	99
	4.2 Grundlegende Klassen	102
	4.3 Application- und Bean-Klassen	115

Kapitel 5	Hinweise zur Programmierung mit dem Portal Manager API	133
5.1	Authentifizierung und Session-Management	133
5.2	Anwendungsbeispiele für das Portal Manager API	140
5.3	Framework für Applications	160
5.4	Gestaltung einer JSP-Seite	166
5.5	Sicherheitsaspekte	168
5.6	Internationalisierung	170
Anhang A	Metadatenschemata	179
Anhang B	Caching	185
B.1	Zugriff auf WCM-Objekte	185
B.2	Deployment-Cache	187
B.3	Funktionsweise des Caches für WCM-Objekte	188
B.4	Cachen von RepositoryEntry-Objekten	189
Glossar		193
Index		197

Abbildungsverzeichnis

Abb. 1 –	Das Schichtenmodell des Portal Manager API	20
Abb. 2 –	Das Schichtenmodell des Portal Manager API mit Session- und Application-Scope	23
Abb. 3 –	Aufbau eines Minimalsystems	28
Abb. 4 –	Verteiltes WCM-System mit separaten Datenbanken	30
Abb. 5 –	Der Lebenszyklus eines WCM-Objekts	47
Abb. 6 –	Prinzipielles Klassendesign im Portal Manager API	51
Abb. 7 –	Die Elemente der Konfiguration	53
Abb. 8 –	Übersicht über die verfügbaren Repositories	55
Abb. 9 –	Repository anlegen	57
Abb. 10 –	Anlegen eines Parameters für ein Repository	58
Abb. 11 –	Zu einem Repository zugeordnete Applications	66
Abb. 12 –	Struktur unterhalb des Elements <i>Repositories</i>	67
Abb. 13 –	Übersicht über die verfügbaren Applications	69
Abb. 14 –	Basisdaten der Application festlegen	71
Abb. 15 –	Assistent beenden?	72
Abb. 16 –	Repository beim Anlegen einer Application zuordnen	73
Abb. 17 –	Ressourcen beim Anlegen einer Application zuordnen	74
Abb. 18 –	Anlegen eines Parameters für eine Application	76
Abb. 19 –	Zu einer Application zugeordnete Repositories	86
Abb. 20 –	Struktur unterhalb des Elements <i>Applications</i>	88
Abb. 21 –	Zu einem Content-Server zugeordnete Applications	89
Abb. 22 –	Das <code>DirectoryRepository</code> mit seinen Parametern	93
Abb. 23 –	Darstellungsschema für die Klassendiagramme	101
Abb. 24 –	Mehrstufige Schlüssel-Wert-Listen in der <code>TupleMap</code>	104
Abb. 25 –	Die grundlegenden Klassen im Gesamtklassendiagramm	105

Abb. 26 –	Die Filterungs- und Sortierungsklassen im Gesamtklassendiagramm	107
Abb. 27 –	Die Klasse <code>RepositoryMap</code> im Gesamtklassendiagramm	107
Abb. 28 –	Strukturelle Beziehungen der komplexen Datentypen	108
Abb. 29 –	Mehrstufige Schlüssel-Wert-Listen in der <code>RepositoryMap</code>	110
Abb. 30 –	Die <code>Repository</code> -Klassen im Gesamtklassendiagramm	114
Abb. 31 –	Die <code>Application</code> -Klassen im Gesamtklassendiagramm	117
Abb. 32 –	Die <code>Bean</code> -Klassen im Gesamtklassendiagramm	121
Abb. 33 –	Aufbau von <code>VipProfile</code> und der darin verwalteten Daten	125
Abb. 34 –	Modus <code>Trusted-Login</code> über den <code>Admin-Client</code> konfigurieren	136
Abb. 35 –	Authentifizierung und <code>Session-Management</code>	138
Abb. 36 –	<code>Login-Repositories</code>	139
Abb. 37 –	Klassendiagramm für <code>Applications</code>	161
Abb. 38 –	Allgemeine Hierarchie zum Entwickeln von <code>Applications</code>	163
Abb. 39 –	Die Klasse <code>RepositoryImpl</code> mit allen für Datenquellen relevanten Methoden	164
Abb. 40 –	<code>JSP-Modell-2-Architektur</code>	166
Abb. 41 –	Architektur des <code>Portal Manager API</code> mit Aspekten der Internationalisierung	171
Abb. 42 –	Struktur einer Website	174

Tabellenverzeichnis

Tabelle 1 –	Verfügbare Funktionalität in den APIs von Livelink WCM Server	14
Tabelle 2 –	Funktionen für die Konfiguration von Repositories und Applications	54
Tabelle 3 –	Repositories und ihre Parameter	60
Tabelle 4 –	WCM-Applications und ihre Parameter	77
Tabelle 5 –	Startparameter für Content-Server, die im Kontext einer JSP-Engine laufen	115
Tabelle 6 –	Beispielkonfiguration eines DirectoryRepository	142

KAPITEL 1

Einleitung

Das Portal Manager API ist Teil von Livelink Web Content Management ServerTM (kurz: Livelink WCM Server). Mit dem Kauf von Livelink WCM Server haben Sie ein leistungsfähiges Content-Management-System erworben, mit dem Sie komplexe und verteilte Websites aufbauen, pflegen und verwalten können.

Mit dem Portal Manager API können Sie dynamische Inhalte für Ihre Website erstellen und verwalten und so Unternehmensportale entwickeln. Darüber hinaus stellt das Portal Manager API die Grundlage für die Personalisierung von Websites dar.

Das Portal Manager API ist in Application-Servern, die der J2EE-Spezifikation genügen, einsetzbar und bietet darüber die Möglichkeit zur Lastenverteilung (dynamic load balancing) und Ausfallsicherheit (session fail over). Außerdem dient das Portal Manager API als Plattform zur Enterprise Application Integration (EAI). Das Portal Manager API bildet die Basis für eine Vielzahl von Connector-Produkten, mit deren Hilfe Sie die Verbindung zu Produkten anderer Hersteller aufbauen können.

Das Portal Manager API ist kein Produkt, mit dem Sie nach der Installation fertige Webportale erhalten. Es ist als Framework zu betrachten, das Sie beim Erstellen von Webportalen und bei den dafür notwendigen Problemlösungen unterstützt. Eine nahtlose Integration in Livelink WCM Server ist dabei garantiert. Das Portal Manager API versetzt Sie in die Lage, schnell in sich konsistente und den Ansprüchen moderner Websites entsprechende, dynamische Komponenten nach Ihren eigenen Vorstellungen zu entwickeln.

Achtung! Das falsche Verwenden der in diesem Handbuch beschriebenen Programmierschnittstelle kann zu Fehlern im WCM-System bis hin zum Absturz des Systems und zu Datenverlusten führen. Durch eine falsche Programmierung können auch Probleme in den Bereichen Performance und Systemressourcen auftreten. Es ist deshalb wichtig, die entwickelte Software erfolgreich hinsichtlich Korrektheit, Stabilität, Robustheit und Performance zu testen, bevor sie in den produktiven Betrieb übernommen wird.

Für das korrekte Funktionieren der entwickelten Software kann keinerlei Garantie durch die Gauss Interprise AG übernommen werden. Gern unterstützen wir Sie beratend und entwickelnd durch den Bereich Gauss Professional Services, um so Probleme bereits in einer frühen Phase der Entwicklung zu verhindern.

1.1 Hinweise zu diesem Handbuch

Das Portal Manager API und damit auch dieses Dokument richten sich an Personen, die über fundiertes technisches Know-how verfügen. Die folgenden Voraussetzungen sollten erfüllt sein:

- Kenntnis des Produkts Livelink WCM Server, möglichst nicht nur als Anwender, sondern als Administrator
- Technisches Verständnis für die Abläufe bei der Benutzung von JSP-Seiten und Servlets
- Allgemeine Programmiererfahrung, Erfahrung in der Programmierung mit Java und Kenntnisse über HTML

Der Inhalt dieses Handbuchs ist folgendermaßen gegliedert:

- Kapitel 2 “Konzepte” erläutert die grundlegenden Konzepte des Portal Manager API und die Integration des Portal Manager API in die Systemarchitektur eines WCM-Systems.
- Kapitel 3 “Portal Manager API verwalten” beschreibt, wie Sie das Portal Manager API mithilfe von Applications und Repositories konfigurieren. Außerdem enthält das Kapitel Hinweise zu LDAP.
- Kapitel 4 “Klassen und Interfaces des Portal Manager API” enthält eine Beschreibung der Programmierschnittstelle, die für die Erstellung von JSP-Seiten bekannt sein muss.
- Kapitel 5 “Hinweise zur Programmierung mit dem Portal Manager API” bietet Ihnen eine Einführung in die Programmierschnittstellen des Portal Manager API sowie einige Anwendungsbeispiele.
- Im Anhang A “Metadatenschemata” sind die Metadatenschemata der Versionen 5e und 8 gegenübergestellt.
- Anhang B “Caching” beschreibt das Caching-Verhalten von des Portal Manager API.

Die hier beschriebene Programmierschnittstelle ist Bestandteil von Livelink WCM Server. Zusätzlich zum vorliegenden Programmierhandbuch können Sie Informationen aus folgenden Quellen beziehen:

- Livelink WCM Server-Installationshandbuch: Dieses Dokument beschreibt die Installation des WCM-Systems und gibt Konfigurationshinweise für Webserver und Application-Server.
- Livelink WCM Server-Administratorhandbuch: Dieses Dokument beschreibt die Konfiguration und Administration eines WCM-Systems aus der Sicht des Administrators.

- WCM Java API-Programmierhandbuch: Dieses Dokument enthält Informationen über Interfaces, Klassen und Methoden der Java-Programmierschnittstelle (WCM Java API), mit der die Funktionalität der WCM-Server über externe Programme genutzt werden kann.
- Javadoc: Hier finden Sie die komplette Dokumentation der Klassen des Portal Manager API.

Es gibt verschiedene flexible Programmierschnittstellen, mit denen die Kernfunktionalitäten von Livelink WCM Server über externe Programme genutzt werden können. Die folgende Tabelle beinhaltet eine Übersicht über die verschiedenen Schnittstellen und die jeweils verfügbare Funktionalität.

Tabelle 1 – Verfügbare Funktionalität in den APIs von Livelink WCM Server

Funktionalität des	WCM Java API	Remote-API	Portal Manager API	WCM WebServices
AdminHandler	✓	✓	✓ nur lesend über VipUserBean	
AttributeHandler	✓	✓	✓ über VipWebsiteBean	✓ über VipWebService_ Port
EventDispatcher	✓	✓		

Funktionalität des	WCM Java API	Remote-API	Portal Manager API	WCM WebServices
MailHandler	✓		✓ über VipEmailBean	
ObjectHandler	✓	✓	✓ über VipObject HandlerBean	✓ über VipWebService_ Port
ObjectLoader	✓	✓		
SystemHandler	✓	✓		
WorkflowHandler	✓		✓ über VipWorkflow Bean	✓ über VipWebService_ Port
DeploymentHandler	✓	✓		
ContextHandler	✓	✓		
LivelinkAdmin Handler	✓		✓ über LivelinkUser Bean	
LivelinkObject Handler	✓		✓ über LivelinkObject Bean	

1.2 Typographische Konventionen

Programmelemente u.Ä. werden im Text folgendermaßen hervorgehoben:

Element	Schriftart oder Symbol	Beispiele
Programmoberfläche wie z.B. Menübefehle, Fenster, Dialoge, Feld- und Schaltflächenbezeichnungen	<i>Menü → Eintrag</i>	<i>Datei → Anlegen</i>
Pfade zu Verzeichnissen, Namen von Dateien und Verzeichnissen	Laufwerk:\Verzeichnis\Dateiname	D:\WCM\admin.bat
Zitate aus Programmcode oder Konfigurationsdateien	Code-Zitate	<head> <title>heading </title> </head>
Variablen, d.h. Platzhalter für bestimmte Elemente	{Variable}	{WCM-Installationsverzeichnis}

Wichtige **Hinweise** und **Warnungen** stehen in grauen Kästen. Diese Informationen sollten Sie unbedingt lesen, um Fehler bei der Nutzung und Verwaltung von WCM-Systemen sowie Datenverluste zu vermeiden.

KAPITEL 2

Konzepte

Dieses Kapitel beschreibt die Einbindung des Portal Manager API in ein WCM-System. Außerdem werden die allgemeinen Konzepte der Programmierung mit dem Portal Manager API dargestellt.

2.1 Architektur des Portal Manager API

Die Funktionalität des Portal Manager API ergibt sich aus dem Zusammenwirken der verschiedenen Komponenten eines WCM-Systems. Dazu gehören insbesondere

- die Server (z.B. Master-Admin-Server, Master-Content-Server, Proxy-Content-Server)
- die im WCM-System verwalteten Websites
- die Session-Beans und anderen Klassen des Portal Manager API
- die JSP-Seiten, die diese Schnittstellen nutzen

Der interne Aufbau des Portal Manager API kann durch ein einfaches Schichtenmodell beschrieben werden (siehe folgende Abbildung).

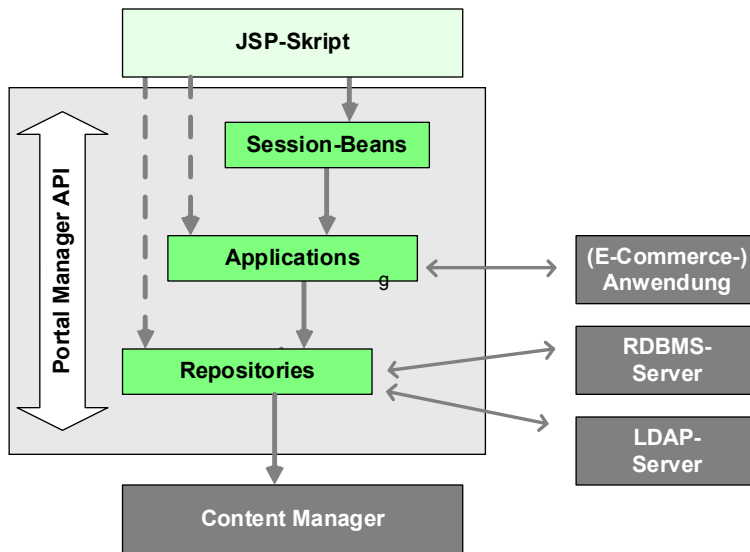


Abb. 1 – Das Schichtenmodell des Portal Manager API

JSP-Skripte

JSP-Skripte sind in HTML-Seiten eingebetteter Java-Code. Durch Anfragen von Clients (HTML-Browsern) an den Webserver wird der Java-Code in solchen HTML-Seiten durch die JSP-Engine ausgeführt. Das Ergebnis wird als Content an den Client zurückgegeben. Der Java-Code wird also serverseitig ausgeführt. Dabei wird der Java-Code mittels einer JSP-Engine einmalig in Servlets umgewandelt, bevor er das erste Mal ausgeführt wird. Der Client bemerkt diesen Vorgang nicht, da ihm nach wie vor HTML-Seiten geliefert werden.

Java-Code, der in JSP-Seiten verwendet wird, muss in Tags eingeschlossen werden, die der JSP-Syntax entsprechen. Andernfalls kann der Code nicht von der JSP-Engine verarbeitet werden.

Session-Beans

Beans sind die Klassen, deren Instanzen in JSP-Skripten benutzt werden können. Sie bilden den Einstiegspunkt in das Portal Manager API. Ein Bean stützt sich auf eine Application, abhängig davon, welche Aufgaben mit dem Portal Manager API gelöst werden sollen. Wie auch bei Applications wird der Anschluss externer Anwendungen erst durch die Entwicklung individueller Bean-Klassen möglich.

Die im Portal Manager API enthaltenen Beans sind jeweils an eine JSP-Session geknüpft. Sie werden daher auch *Session-Beans* genannt.

Sessions sind Einheiten, die von der JSP-Engine verwaltet werden. Der Sinn einer Session besteht darin, Aktionen eines Webclients zusammenfassen zu können. Dieses ist mit HTTP als Protokoll zwischen Webclient (HTML-Browser) und Webserver zunächst einmal unmöglich, da es sich um ein zustandsloses Protokoll handelt. Eine Session existiert dagegen auch HTTP-Anfragen-übergreifend. Ein JSP-Skript kann die aktuelle Session referenzieren. So lässt sich ein Zustand für jeden Benutzer über mehrere Anfragen hinweg verwalten. Durch Session-Beans wird die Ressourcenverwaltung erheblich erleichtert.

Der Beginn einer Session ist durch den ersten Kontakt eines Webclients festgelegt, das Ende durch den letzten Kontakt. Technisch ist die JSP-Engine für die Verwaltung der Sessions verantwortlich. Sie speichert dazu im Allgemeinen einen Session-Schlüssel im jeweiligen Browser. Dies ist auch der Grund dafür, dass im Browser (nicht-persistente) Cookies zugelassen sein müssen. Jede JSP-Engine hat eigene Strategien, um zu bestimmen, wie und wann Session-Schlüssel ungültig werden, wann also der letzte Kontakt eines Browsers stattfand. Im Normalfall wird dies empirisch über ein Zeitverhalten bestimmt. Verschiedene Instanzen von Browsern auf demselben Client-Computer sind normalerweise mit verschiedenen Session-Schlüsseln assoziiert. Der gesamte Komplex der JSP-Engines ist weitgehend standardisiert.

Applications

Applications sind Beschreibungen beliebiger Anwendungen, auf die im Portal Manager API Bezug genommen werden soll. Das Portal Manager API umfasst bereits einige Implementationen von Applications, die den internen Zugriff auf Livelink WCM Server regeln. Allerdings müssen weitere, individuelle Implementationen entwickelt werden, wenn externe Anwendungen wie etwa E-Commerce-Lösungen über das Portal Manager API gesteuert werden sollen.

Applications ermöglichen den Zugriff auf Datenquellen. Das im Portal Manager API vorgesehene Konzept für das Verarbeiten des Datenbestandes einer Application basiert auf dem Einsatz von Repositories. Repositories sind durch die grundlegenden Klassen modelliert, die in Kapitel 4 "Klassen und Interfaces des Portal Manager API" beschrieben sind.

Unter Applications wird im Portal Manager API also eine Sammlung von im weitesten Sinne abstrakten Zugriffsverfahren verstanden, mit der die Sicht auf beliebige Anwendungen vereinheitlicht wird. Neue, konkrete Applications können unter Ausnutzung der Application-Schnittstelle besonders einfach integriert werden. Die Benutzung dieser Schnittstelle ist zwar nicht obligatorisch, Sie erhalten darüber aber den vollen Funktionsumfang der Session-Beans.

Im Unterschied zu Session-Beans ist die Lebenszeit der Ressourcen in Applications unbegrenzt. Applications kennen keine Sessions. Es gibt nur einen einzigen Ressourcenkontext, der mit dem Laufzeitbeginn der Application anfängt und mit dem Laufzeitende aufhört. Die Application wird bei Bedarf von dem Bean instanziiert und gegebenenfalls durch den serverseitigen Prozess (Java Virtual Machine) gestoppt.

Die folgende Abbildung zeigt, wo die Grenze zwischen Session- und Application-Scope liegt.

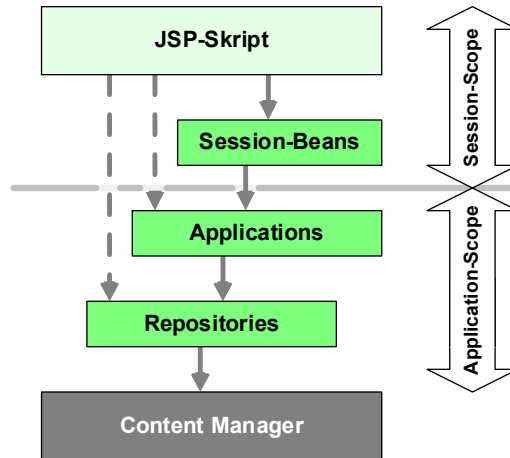


Abb. 2 – Das Schichtenmodell des Portal Manager API mit Session- und Application-Scope

Repositories

Repositories sind – ähnlich wie Applications – eine Abstraktion, die einheitliche Zugriffe auf Datenbestände erlaubt. Neue Anwendungen, die unter Ausnutzung der Application-Schnittstelle erstellt werden, können auf bestehende oder neu zu entwickelnde Repositories zugreifen.

2.2 Integration des Portal Manager API in das WCM-System

Die Integration des Portal Manager API in das WCM-System kann unter verschiedenen Gesichtspunkten betrachtet werden:

- Zugriff von Portal Manager API-Lösungen auf Daten des WCM-Systems
- Einbinden von Content-Servern, auf denen das Portal Manager API zur Verfügung steht, in das WCM-System

Zugriff auf Daten des WCM-Systems

Aus Sicht der Lösungen, die auf dem Portal Manager API basieren, sind das Portal Manager API und WCM-System über die Repository-Schnittstelle miteinander gekoppelt: Wenn Sie eine Anwendung mit dem Portal Manager API programmieren, verwenden Sie Beans, um auf die Daten des WCM-Systems zuzugreifen. Das Bean fordert das benötigte Repository von der Application an und stellt die Verbindung zum WCM-System über dieses Repository her. Über das Repository erhalten Sie dann die Objektdaten aus dem WCM-System für die Portal Manager API-Anwendung. Die Zuordnung zwischen Repositories und Applications nehmen Sie über den Admin-Client vor, siehe "Repository zu einer Application zuordnen" auf Seite 86.

Es gibt mehrere Beans, die bestimmte Services im WCM-System ansprechen. Über sie kann auf Daten der verwalteten Objekte ebenso zugegriffen werden wie auf administrative Daten. Hierdurch wird der Zugriff auf Objekthinhalte, Objektmetadaten, Benutzer-, Gruppen-, Rollen- und Funktionsbereichsdaten ermöglicht. Auch die Filterung und Sortierung über den Datenbestand des WCM-Systems wird unterstützt.

Portal Manager API und Deploymentsysteme

Beim Zugriff einer Anwendung auf Objektdaten über ein Repository wird immer auf die Daten eines bestimmten Deploymentsystems zugegriffen. In der Konfiguration jedes Repository, über das auf Objekthinhalte und Metadaten zugegriffen werden kann, ist ein Deploymentsystem angegeben. Wenn Sie über den Admin-Client eine neue Website anlegen, werden die Repositories für alle Deploymentsysteme der neuen Website automatisch erzeugt.

Das Portal Manager API kann auf die Daten beliebiger Deploymentsysteme zugreifen. Die einzige Voraussetzung dabei ist, dass die entsprechenden Deploymentsysteme dieselbe Datenhaltung verwenden wie der Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft. Für den Zugriff auf Objekte mit WebDAV-fähigen Clients wird ein Servlet verwendet, das das Portal Manager API voraussetzt.

Portal Manager API und Datenhaltungen

Das Portal Manager API benötigt sowohl Zugriff auf die Daten von Deploymentsystemen (z. B. die generierten URLs von Objekten) als auch auf die WCM-Objekte selbst (über Repositories). Der Zugriff auf die WCM-Objekte ist erforderlich, um z. B. die Metadaten zu lesen und für Anwendungen zu verarbeiten. Um auf die Daten von Deploymentsystemen zugreifen zu können, muss das Deploymentsystem des Repository auf einem Content-Server installiert sein, der dieselbe Datenhaltung verwendet wie der Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft.

In einem verteilten WCM-System ist es möglich, dass z. B. zwei Edit-Deploymentsysteme für dieselbe Website auf unterschiedlichen Servern installiert wurden. Wenn diese beiden Server über separate Datenhaltungen verfügen, werden auch bei der Generierung der Seiten durch die Deploymentsysteme verschiedene Datenhaltungen benutzt. Das Portal Manager API kann nicht auf beide Edit-Deploymentsysteme

zugreifen, da dazu der gleichzeitige Zugriff auf die Daten *einer* Website über verschiedene JDBC-Pools erforderlich wäre. In einem solchen Fall sind zwei Content-Server erforderlich, die im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server laufen.

Das Portal Manager API in ein WCM-System einbinden

Die Server werden in die zwei Hauptkategorien **Master** und **Proxy** unterteilt. In jedem WCM-System gibt es einen Master-Administrationsserver und einen bzw. mehrere Master-Content-Server.

Der **Master-Administrationsserver** steuert die Benutzerverwaltung und übernimmt Konfiguration, System- und Lizenzverwaltung. Der Zugang zum Administrationsserver erfolgt über den Admin-Client.

Der **Master-Content-Server** verwaltet eine oder mehrere Websites, wobei eine Website immer genau einem Master-Content-Server zugeordnet ist. Nur an einem Master-Content-Server können Änderungen an Inhalt und Status von WCM-Objekten durchgeführt werden. Auf dem Master-Content-Server stehen immer alle Datenhaltungssichten (Edit, QS und Produktion) zur Verfügung.

Zusätzlich zum Master-Content-Server können Sie in einem WCM-System **Proxy-Content-Server** einrichten, auf denen die Daten der Website(s) mithilfe entsprechender Deploymentsysteme ebenfalls verfügbar gemacht werden. Im Unterschied zu einem Master-Content-Server haben Proxy-Content-Server jedoch nur lesenden Zugriff auf die Daten. Sollen Website-Daten über einen Proxy-Content-Server bearbeitet werden, wendet sich der Proxy-Content-Server an den Master-Content-Server. Dieser sperrt das Objekt für weitere schreibende Zugriffe und speichert nach Abschluss der Bearbeitung die geänderten Objekte in der Datenhaltung. Anschließend informiert der Master-Content-Server alle angeschlossenen Proxy-Content-Server der Website darüber, dass sich

das WCM-Objekt geändert hat. Auf diese Weise bleiben Ihre Website-Inhalte konsistent.

Für die Verteilung der generierten Seiten im WCM-System sind die so genannten **Deploymentsysteme** zuständig.

Das Portal Manager API steht auf allen Content-Servern zur Verfügung, die im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server laufen. Für die Nutzung des Content-Clients und des Content-Clients (Classic) ist ein derartiger Content-Server erforderlich.

Bitte beachten Sie für das Einrichten von Content-Servern, die im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server laufen, folgende Hinweise:

- Der Zugriff auf die Deploymentsysteme einer Website, die unterschiedliche Datenhaltungen verwenden, muss über verschiedene Content-Server erfolgen.
- Die verfügbaren Sichten auf die Daten einer Website und mögliche Aktionen hängen von den Routing-Einstellungen des Content-Servers ab. Sollen über den Content-Server Änderungen an den Daten einer Website durchgeführt werden, z. B. über den Content-Client, muss der Content-Server die Edit-Sicht der Daten erhalten (in diesem Fall hat der Content-Server Zugriff auf alle drei Sichten der Website).

Aufgrund der flexiblen Systemarchitektur von Livelink WCM Server gibt es für den Aufbau eines WCM-Systems zahlreiche Möglichkeiten. Das Livelink WCM Server-Administratorhandbuch enthält ausführliche Informationen zu diesem Thema.

In den folgenden Abschnitten werden zwei Szenarien für die Integration des Portal Manager API in ein WCM-System dargestellt.

Minimalsystem

Das WCM-System besteht aus einem Master-Admin-Server und einem Master-Content-Server. Der Master-Content-Server wird im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server ausgeführt. Auf dem Master-Content-Server steht damit das für die Bearbeitung von Websites mit dem Content-Client erforderliche Portal Manager API zur Verfügung.

Die Website "InternetSite" wird auf dem Master-Content-Server angelegt. Die Objekte der Website werden über drei Deploymentsysteme (Edit, QS und Produktion) in unterschiedliche Verzeichnisse verteilt. Damit wird auf dem Master-Content-Server das komplette Staging realisiert. Für "Deploymentsystem" wird in der Abbildung die Abkürzung "DS" verwendet.

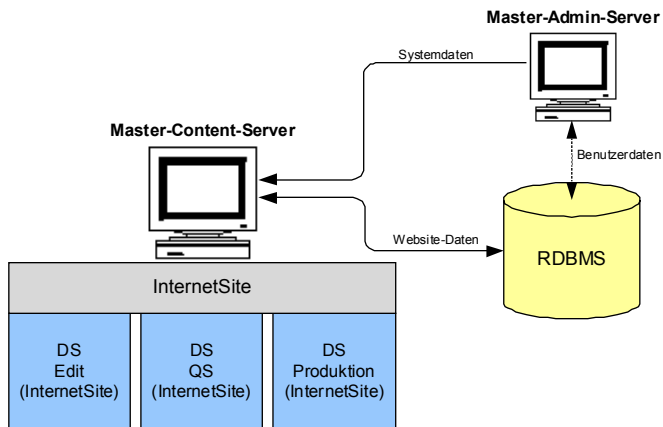


Abb. 3 – Aufbau eines Minimalsystems

Ein Minimalsystem installieren Sie mithilfe der entsprechenden Option im WCM-Installationsprogramm (siehe Livelink WCM Server-Installationshandbuch). Um eine Website in diesem Szenario anzulegen, starten Sie den Assistenten für neue Websites im Admin-Client, und legen Ihre Website mit der Option *Minimal* bzw. *Minimal (dynamisch)* an.

Verteiltes WCM-System mit separaten Datenhaltungen

Die flexible und skalierbare Systemarchitektur von Livelink WCM Server ermöglicht Ihnen den Aufbau verteilter WCM-Systeme entsprechend Ihrer Unternehmensstruktur. Um die Datenübertragung zwischen den einzelnen Servern zu minimieren, können die Proxy-Server auf separate Datenhaltungen zugreifen.

Bei der Nutzung des Portal Manager API in einem solchen Szenario ist zu beachten, dass ein Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft, nicht gleichzeitig auf unterschiedliche Datenhaltungen zugreifen kann (siehe auch "Portal Manager API und Datenhaltungen" auf Seite 25). Werden die Daten Ihrer Website über verschiedene Proxy-Content-Server gepflegt, die auf separate Datenhaltungen zugreifen, müssen diese Server jeweils im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server laufen.

Das folgende Beispiel soll ein solches Szenario verdeutlichen:

Das WCM-System besteht aus einem Master-Admin-Server und einem Master-Content-Server. Die Website "InternetSite" wird im WCM-System angelegt. Das System ist auf drei weitere Proxy-Content-Server verteilt. Zwei dieser Proxy-Content-Server laufen im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server – über sie erfolgt die Bearbeitung der WCM-Objekte mithilfe des Content-Clients. Auf diesen beiden Servern sind daher entsprechende Edit- und QS-Deploymentssysteme installiert. Der dritte Proxy-Content-Server dient zur Veröffentlichung der Produktionssicht der Website, auf ihm ist ein Produktionsdeploymentssystem eingerichtet.

Um die Darstellung übersichtlicher zu gestalten, sind die Verbindungen vom Master-Admin-Server zu den anderen Servern in der folgenden Abbildung nicht enthalten.

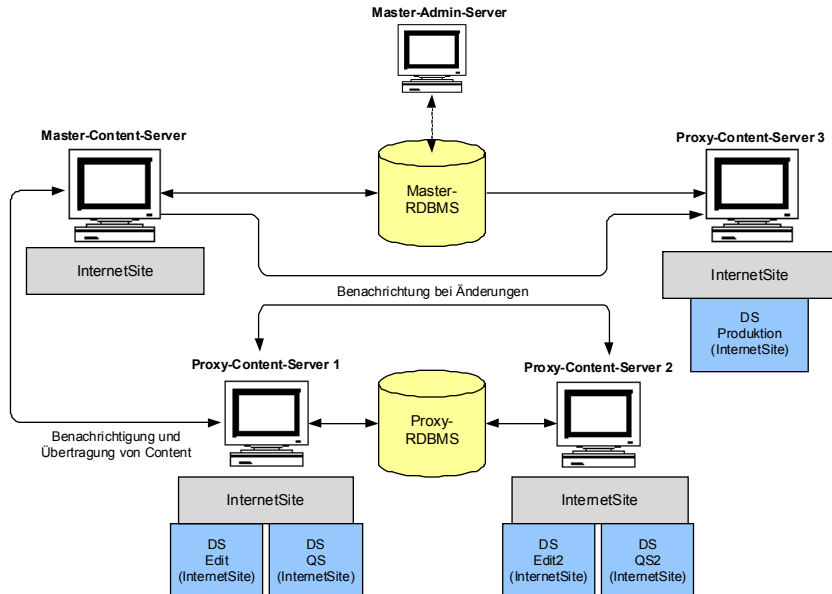


Abb. 4 – Verteiltes WCM-System mit separaten Datenbanken

Wird eine Änderung an einem WCM-Objekt vorgenommen, benachrichtigt der Master-Content-Server die Proxy-Content-Server 1 und 3. Proxy-Content-Server 1 benachrichtigt Proxy-Content-Server 2 über die Änderungen. Auf allen benachrichtigten Servern startet das Deployment für die geänderten Objekte, sodass die entsprechenden Seiten aktualisiert werden.

Ein solches verteiltes System richten Sie mithilfe der Option *Benutzerdefinierte Installation* im WCM-Installationsprogramm ein (siehe Livelink WCM Server-Installationshandbuch).

Um eine Website in diesem Szenario anzulegen, starten Sie den Assistenten für neue Websites im Admin-Client und legen Ihre Website mit der Option *Benutzerdefiniert* an.

Content-Server im Application-Server starten

Sowohl der Content-Client als auch der Content-Client (Classic) verwenden das Portal Manager API. Um dieses API – auch für selbst erstellte JSP-Seiten – zur Verfügung zu stellen, ist es erforderlich, einen Content-Server im Kontext des Application-Servers zu starten.

Content-Server als Webanwendung

Wenn für den Content-Server eine Webanwendung erzeugt wurde, läuft der Content-Server im Application-Server. Unterstützt der verwendete Application-Server den Servlet-Standard 2.3, wird der Content-Server automatisch zusammen mit der entsprechenden Webanwendung gestartet und gestoppt.

Unterstützt der Application-Server den Servlet-Standard 2.2, werden die Webanwendung und der Content-Server getrennt gestartet und gestoppt. Verwenden Sie in diesem Fall die entsprechenden Skripte für den Content-Server (siehe folgenden Abschnitt). Achten Sie darauf, dass die Webanwendung zuerst gestartet wird.

Content-Server in JSP-Engine starten

Wenn der Content-Server im Kontext einer JSP-Engine läuft, die keine Webanwendungen unterstützt, müssen zum Starten des Servers die im Verzeichnis **{WCM-Installationsverzeichnis}\tools** befindlichen Skripte verwendet werden.

Folgende Schritte sind dazu erforderlich:

1. Um den für den Content-Server benötigten Klassenpfad zu setzen, rufen Sie das Skript **setPomaClasspath.bat** bzw. **.sh** auf. Dieses Skript befindet sich im Verzeichnis **{WCM-Installationsverzeichnis}\tools**.

2. Fügen Sie den Klassenpfad, der durch das Aufrufen des Skripts erstellt wurde, dem Klassenpfad der JSP-Engine hinzu.
3. Kopieren Sie das mitgelieferte Skript **portalmanager.bat** bzw. **.sh** aus dem Verzeichnis **{WCM-Installationsverzeichnis}\tools** in das Wurzelverzeichnis der WCM-Installation; benennen Sie das Skript um in **{Name des Content-Servers}.bat** bzw. **.sh**.
4. Ersetzen Sie im Skript **{Name des Content-Servers}.bat** bzw. **.sh** alle Platzhalter **SERVERNAME** durch den Namen des Content-Servers.

Damit der Content-Server über das Skript gestartet werden kann, muss die JSP-Engine bereits laufen. In der Konfiguration der verwendeten JSP-Engine muss das Servlet-Mapping `/servlet/*` eingetragen sein.

Hinweis: Wenn Sie mehrere Content-Server auf demselben Rechner einrichten, müssen unterschiedliche JSP-Engines zum Ausführen der Server verwendet werden. Passen Sie in diesem Fall nach der Installation des WCM-Systems in den Skripten zum Starten der Content-Server die eingerichtete Standard-URL entsprechend der Konfiguration der verwendeten JSP-Engine an.

Zum Beenden des Content-Servers verwenden Sie das mitgelieferte Skript **shutdown_{Name des Content-Servers}.bat** bzw. **.sh**. Alternativ kann der Server auch über den Admin-Client heruntergefahren werden.

2.3 Programmierkonzepte

Der Einsatz des Portal Manager API zur Dynamisierung und Personalisierung von Websites setzt voraus, dass die grundlegenden Programmierkonzepte des Portal Manager API bekannt sind. Sie werden Ihnen in diesem Abschnitt vorgestellt.

Mit dem Portal Manager API können Sie Programme für die JSP-Engine, so genannte JSP-Skripte, entwickeln. JSP-Skripte stellen den zum Benutzer gewandten Teil einer Website bzw. eines Portals dar, vergleichbar mit der GUI einer konventionellen Anwendung. Darüber hinaus bietet das Portal Manager API aber auch weit reichende Unterstützung bei der Integration neuer Anwendungen.

Für die Erstellung von JSP-Skripten wie auch für die Erstellung von Anwendungen werden Schnittstellen in Form von fertigen Java-Klassen und JavaBeans mitgeliefert. Die Java-Klassen und JavaBeans setzen die in diesem Kapitel vorgestellten Konzepte um. Einen Überblick über die Klassen und Beans vermittelt Ihnen Kapitel 4 "Klassen und Interfaces des Portal Manager API", detaillierte Informationen entnehmen Sie bitte der Javadoc zum Portal Manager API.

Dynamischer Content

Das Portal Manager API eignet sich für die Erstellung und Verwaltung dynamischer Inhalte einer Website (so genannter *Content*), die charakteristisch für Unternehmensportale sind. Im Gegensatz zu statischem Content, der schon während der Erstellung durch einen Bearbeiter oder Autor festliegt, wird der Inhalt von dynamischem Content komplett oder in Teilen erst zur Laufzeit erzeugt. Es besteht so die Möglichkeit, auf die aktuelle Situation und den aktuellen Kontext während der Nutzung der Website zu reagieren.

Zwar stellt das in Livelink WCM Server umgesetzte Vorlagenkonzept im Ansatz auch einen Schritt zur Dynamisierung dar, da der Inhalt durch die Verbindung zweier oder mehrerer Content-Objekte erstellt wird. Das Ergebnis ist jedoch wiederum statisch, passt sich also dem aktuellen Einsatzkontext nicht mehr an.

“Echte” Dynamisierung ermöglicht dagegen die Kontrolle des Contents auch zur Laufzeit, also in dem Moment, in dem eine Seite von einem Benutzer angefordert wird. Die Mächtigkeit lässt sich an besonders typischen Anforderungen moderner Websites beschreiben. Diese werden im Folgenden anhand einer Auswahl häufiger und typischer Einsatzfälle des Portal Manager API vorgestellt. Die Szenarien sind als Beispiele zu verstehen, der Einsatz des Portal Manager API ist keinesfalls auf die genannten Beispiele beschränkt.

Navigationselemente

Unter Navigation wird zunächst einmal die Möglichkeit verstanden, die Einordnung der aktuellen Seite in ihren Kontext zu visualisieren, damit der Benutzer den Kontext schnell erfassen kann. Darüber hinaus ermöglicht die Navigation einem Benutzer durch Hyperlinks einen schnellen, unkomplizierten Wechsel zu assoziiertem Content, etwa zu themenverwandten Seiten oder zu Seiten, die in der hierarchischen Struktur benachbart sind. Auch *Sitemaps* gehören zu den Navigationselementen, sie bereiten jedoch im Allgemeinen nicht den Kontext der aktuellen Seite, sondern Informationen über die gesamte Website auf.

Navigationselemente können auf sehr unterschiedliche Art in einer Seite auftreten. Denkbar sind etwa animierte Bestandteile, die Navigationselemente als Menüs oder aufklappbare Baumstruktur präsentieren. Natürlich können Navigationselemente auch mit dem “Look-and-Feel” einer Website gestaltet werden, damit sie sich in das Gesamtkonzept der Website einfügen.

Im WCM-System stehen allein durch die Speicherung von Metadaten zu jedem Content-Objekt schon genügend Informationen für eine Navigation zur Verfügung. So werden zu jedem Content-Objekt diejenigen Objekte gespeichert, die per Hyperlink erreichbar sind, sowie die Objekte, von denen aus das aktuelle Content-Objekt per Hyperlink erreichbar ist. Weitere Informationen für eine Navigation sind dort durch den Themenbaum und die eventuelle Verknüpfung eines Content-Objekts mit einem Vorlagenobjekt gegeben.

Wenn sich die zugrunde liegenden Metadaten ändern, ändert sich der Inhalt automatisch, ohne dass manuelle Anpassungen nötig sind. Löschen, Kopieren, Verschieben von Objekten führt dann nicht mehr dazu, dass die Navigationselemente veralten.

Personalisierung

Personalisierung bedeutet, dass die Inhalte der Website individuell – abhängig vom aktuellen Benutzer – aufbereitet und dargestellt werden. Dieser benutzerspezifische Content kann aus verschiedenen Gründen gewünscht sein.

Möglicherweise ist es nötig, bestimmten Benutzern definierte Informationen vorzuenthalten. Ein statisches System wie Livelink WCM Server kann zwar Benutzern ohne ausreichende Rechte Content vorenthalten, allerdings wird dadurch nicht verhindert, dass der Benutzer Kenntnis über die Existenz bestimmter Objekte hat, für die er keine Rechte besitzt.

Eine andere Einsatzmöglichkeit personalisierter Seiten bezieht sich auf vom Benutzer definierte, individuelle Einstellungen, Vorlieben etc. oder seinen aktuellen Kontext. Ein typisches Beispiel hierfür ist ein E-Commerce-Warenkorb-System: Hier wird dem Benutzer immer nur sein persönlicher aktueller Warenkorb angezeigt.

Personalisierung und Navigation sollten immer Hand in Hand gehen: Die Navigation sollte keinen Content anzeigen, den der Benutzer nicht sehen will oder darf.

Anwendungsintegration

Anwendungen in Websites sind immer dynamische Komponenten. Bei der Integration einer Anwendung in eine Website kommt es darauf an, die Schnittstelle zum Benutzer hinsichtlich der Daten und des “Look-and-Feel” an die Website anzupassen. Im Idealfall werden die Daten einer Anwendung als Content oder als Teile von Content interpretiert, sodass für die Präsentation einer Website die Herkunft des Contents keine Rolle mehr spielt. Die Lösung dieser Aufgabe hängt stark davon ab, welche Schnittstellen die Anwendung anbietet. Das Portal Manager API unterstützt Sie bei der Programmierung und ermöglicht eine solche Integration auf Funktions- und Datenebene.

Platzierung mehrerer Content-Objekte auf einer Seite (Multicontent-Seite)

Mithilfe des Portal Manager API können beliebig viele Content-Objekte auf einer Seite präsentiert werden, wobei einschränkende Aspekte – wie die unter dem Stichwort Personalisierung genannten – natürlich berücksichtigt werden können. Als Content können Objekte aus WCM-verwalteten Website ebenso benutzt werden wie Daten aus Anwendungen. Sogar Website- und anwendungsübergreifende Content-Verknüpfungen stellen kein Problem dar.

Formulare

Durch die Möglichkeiten des Portal Manager API können Sie auf sehr einfache Weise formularbasierte Anwendungen entwickeln. Über formularbasierte Schnittstellen lassen sich beispielsweise Objekte in einer mit Livelink WCM Server verwalteten Website anlegen und bearbeiten.

Unter dem Begriff *Formular* versteht man eine JSP-Seite, die eine formularbasierte Schnittstelle zum Anwender bietet. Über ein solches Formular werden Daten für eine Formularinstanz geholt und bearbeitet.

Die Möglichkeiten der formularbasierten Arbeit werden durch zwei besondere Objekttypen repräsentiert: *Formularvorlage* und *Formularinstanz*.

- Formularvorlage – Vorlage, die den Typ “Formular” definiert
- Formularinstanz – Inhalte eines Formulars (einer Formularvorlage) in Form von XML-Daten

Um auf der Grundlage dieser Objekttypen eine formularbasierte Schnittstelle anzubieten, müssen außerdem die eigentlichen Formulare als JSP-Seiten mit entsprechenden Auswahl- bzw. Eingabemöglichkeiten entwickelt werden.

Die Formularvorlage dient im Wesentlichen nur zum Identifizieren und Typisieren der unterschiedlichen Formularinstanzen. Die Formularinstanz (das Exemplar eines ausgefüllten Formulars) ist eine XML-Datei, die eine bestimmte Formularvorlage als WCM-Vorlage haben kann.

Hinweis: Für die Bearbeitung von XML-Datenstrukturen stehen außerdem die vier Objekttypen “XML-Dokument”, “XML-Vorlage”, “XSLT-Dokument” und “XSLT-Vorlage” zur Verfügung. Ausführliche Informationen zu diesen Objekttypen finden Sie im Content-Client-Benutzerhandbuch.

Verarbeitung von Formulardaten

In einer Formularinstanz werden die Daten der einzelnen Formularfelder in einer XML-Notation abgelegt. Zur Generierung dieser XML-Daten aus einem HTML-Formular werden die Methoden der Klasse `FormData` eingesetzt.

Die Klasse `FormData`

Vollständiger Name:

`de.gauss.vip.portalmanager.util.forms.FormData`

Die Klasse `FormData` bietet Methoden zur Verarbeitung von Formulardaten, insbesondere zum Lesen von Daten aus einem HTML-Formular und Speichern der Formulardaten als XML-Dokument.

Von besonderer Bedeutung ist in diesem Zusammenhang die Methode `processFilledForm` der Klasse `FormData`. Diese Methode erzeugt ein XML-Dokument und extrahiert die Metadaten für das WCM-Objekt aus den übergebenen Formulardaten. Weitere Informationen zu dieser Methode finden Sie in der Javadoc zum Portal Manager API.

Eine weitere Hilfsklasse zur Verarbeitung von Formularen und ihren Daten ist die Klasse `XmlContentConverter`, die das direkte Separieren von Feldern aus dem XML-Dokument erlaubt. Weitere Angaben zu dieser Klasse und ihren Methoden finden Sie in der Javadoc zum Portal Manager API.

Anlegen eines Objekts

Nachdem der Content und die Metadaten für eine Formularinstanz über die Methode `processFilledForm` extrahiert und konvertiert wurden, muss die Formularinstanz mit den entsprechenden Daten angelegt werden. Die Verbindung zu den Funktionen von Livelink WCM Server wird durch die Klasse `VipObjectHandlerBean` hergestellt (siehe Abschnitt "Die Klasse `VipObjectHandlerBean`" auf Seite 128). Informationen zu den Methoden der Klasse finden Sie in der Javadoc zum Portal Manager API.

Das `VipObjectHandlerBean` bildet auch die Basis für ein formularunterstütztes Arbeiten mit den Objekten einer WCM-verwalteten Website. Es besteht also prinzipiell die Möglichkeit, eine HTML-basierte Oberfläche für Livelink WCM Server zu entwickeln. Durch die Methoden des `VipObjectHandlerBean` können über eine solche formularbasierte Oberfläche alle Funktionen vom Anlegen eines neuen Objekts, über Vorlegen zur Qualitätssicherung bis hin zur Freigabe des Objekts abgebildet werden.

Statifizierung

Um den Zugriff auf dynamische Inhalte zu beschleunigen, ist es unter Umständen sinnvoll, den Code der dynamischen Seite in statischen Code umzuwandeln. Mit dem Begriff *Statifizierung* ist die Darstellung von dynamischen Inhalten auf statischen Seiten gemeint. Bei der Statifizierung wird der Code der dynamischen Seite bereits während der (asynchronen) Generierung der Seite ausgeführt und das Ergebnis, im Allgemeinen HTML-Code, gespeichert.

Durch das Statifizieren von Seiten wird ein bestimmter Zustand einmal festgehalten und in der Produktionssicht einer Website publiziert. Bei einer Seite, die beispielsweise Navigationselemente enthält, wird der aktuelle Stand der Navigation zum Zeitpunkt des Publizierens in Form von statischen Elementen in der Seite festgehalten. Änderungen in der Themenstruktur, die nachträglich freigegeben werden, sind in der statifizierten Navigation nicht enthalten. Die dynamischen Elemente der Seite werden erst dann aktualisiert, wenn die Seite neu generiert wird. Für personalisierte Seiten ist die Statifizierung grundsätzlich ungeeignet, da derartige Seiten dynamisch, d.h. zur Laufzeit, auf die eingegebenen Benutzerdaten reagieren müssen.

Die Statifizierung von Seiten hängt von verschiedenen Bedingungen ab.

Konfiguration des Deploymentsystems

- Im Admin-Client wird festgelegt, ob die Statifizierung für alle oder keine Objekte der Website bzw. abhängig vom Objekt durchgeführt wird.
- Im Admin-Client kann die Option *Statifizierte Seiten analysieren* aktiviert werden. Bei dieser Analyse werden die OIDs zu den WCM-URLs, die auf den Seiten verwendet werden, intern vom Content-Server gespeichert. Ändert sich die URL eines referenzierten Objekts, werden die entsprechenden Seiten automatisch neu erzeugt und statifiziert.

Ausführliche Informationen zur Konfiguration von Deploymentsystemen finden Sie in der Online-Hilfe des Admin-Clients.

Objektyp

Es können nur Objekte statifiziert werden, deren Objektyp mit der Attributmenge "dynamic" verknüpft ist. Dies sind die folgenden Objekttypen:

- "JSP-Seite", "JSP-Thema" und "JSP-Vorlage"
- "Formularinstanz" und "Formularvorlage"
- "XML-Dokument", "XML-Vorlage", "XSLT-Dokument" und "XSLT-Vorlage"

Spezialattribute

Folgende Spezialattribute spielen bei der Statifizierung von Objekten eine Rolle:

- **generate_static**
Mithilfe des Spezialattributs `generate_static` wird die objektabhängige Statifizierung gesteuert (siehe Abschnitt "Objektabhängige Statifizierung" auf Seite 42).

Dieses Attribut heißt im Content-Client "Seite statifizieren".

- **timeout**
Wenn der Code einer dynamischen Seite in statischen Code umgewandelt werden soll, wird die betreffende Seite zunächst als temporäre Seite erzeugt. Diese temporäre Seite wird über eine URL-Verbindung ausgeführt. Das Ergebnis dieser Ausführung wird als endgültige statische Seite geschrieben.

Im Admin-Client wird in der Konfiguration des Deploymentsystems die Wartezeit (in Millisekunden) für den Aufruf der URL-Verbindung zu der temporär generierten dynamischen Seite eingestellt. Diesen Wert können Sie mithilfe des Spezialattributs `timeout` im Metadaten-Dialog des Content-Clients überschreiben.

Der für eine Seite gültige Timeout-Wert wird auf die folgende Weise bestimmt:

1. Der Wert des Spezialattributs `timeout` wird für das zu statifizierende Objekt bestimmt.
2. Der Wert des Spezialattributs `timeout` wird für die Vorlage, die dem zu statifizierenden Objekt zugewiesen ist, bestimmt (ggf. wird dazu die Vorlagenkaskade durchsucht).
3. Der Timeout-Wert wird aus der Konfiguration des Deployment-systems gelesen.

Es wird also nur dann der im Admin-Client festgelegte Timeout-Wert verwendet, wenn weder für das zu statifizierende Objekt noch für eine seiner Vorlagen das Spezialattribut `timeout` einen gültigen Wert hat.

Hinweis: Die Statifizierung wird mithilfe des Content-Servers durchgeführt, der im Kontext der JSP-Engine bzw. als Webanwendung im Application-Server läuft. Insbesondere bei Überlastung dieses Content-Servers kann es zur Überschreitung des zulässigen Timeout-Werts und damit verbunden zum Abbruch der Statifizierungsversuche und zu Fehlermeldungen kommen. Wählen Sie deshalb die Timeout-Werte entsprechend großzügig.

■ `absolute_urls`

Das Spezialattribut `absolute_urls` spielt u.a. bei der Statifizierung von Formularinstanzen eine besondere Rolle (siehe Abschnitt "Statifizierung von Formularinstanzen" auf Seite 43).

Objektabhängige Statifizierung

Ein Objekt, dessen Typ mit der Attributmenge “dynamic” verknüpft ist, verfügt über das Spezialattribut `generate_static`. Mithilfe dieses Attributs können Sie im Metadaten-Dialog des Content-Clients einstellen, wie der Code des Objekts und – sofern vorhanden – seiner Vorlage(n) statifiziert werden soll. Voraussetzung dafür ist, dass im Admin-Client für das Deploymentsystem objektabhängige Statifizierung eingestellt ist.

Statifizierung von JSP-Objekten

Ein JSP- Objekt wird objektabhängig statifiziert, wenn das Spezialattribut `generate_static` den Wert `on` oder `local` hat:

- `on`: Dieser Wert bewirkt, dass der Inhalt des Objekts in statischen Code umgewandelt wird. Das gleiche gilt für sämtliche Vorlagen, die dem Objekt zugeordnet sind, soweit ihr Inhalt in die vom Deploymentsystem erzeugte Seite des Objekts eingefügt wird.
- `local`: Dieser Wert bewirkt hinsichtlich des Body-Bereichs, dass nur der Inhalt des Objekts in statischen Code umgewandelt wird. Außerdem wird der Inhalt des gesamten Head-Bereichs, der gegebenenfalls aus den Inhalten der Head-Bereiche der Vorlagenkaskade zusammengesetzt ist, in statischen Code umgewandelt.

Variablen, die in einer JSP-Vorlage definiert sind, können nur von Objekten genutzt werden, die den gleichen “Statifizierungsstatus” wie die JSP-Vorlage haben. Damit das Objekt die Variable aus der JSP-Vorlage nutzen kann, bedeutet dies für eine nicht vollständig statifizierte Vorlagenkaskade, dass

- entweder sowohl der Code der JSP-Vorlage als auch der Code des Objekts statifiziert sein muss
- oder weder der Code der JSP-Vorlage noch der Code des Objekts statifiziert sein darf.

Statifizierung von Formularinstanzen

Formularinstanzen können nur objektabhängig statifiziert werden. Wurde im Admin-Client für ein Deploymentsystem die Statifizierungsoption “immer” eingestellt, können in diesem Deploymentsystem keine Formularinstanzen mithilfe einer JSP-Seite neu angelegt, bearbeitet oder statifiziert werden. In einem derartig konfigurierten Deploymentsystem kann mit Formularen nicht sinnvoll gearbeitet werden.

Obwohl für die Statifizierung von Formularinstanzen auch das Spezialattribut `generate_static` ausgewertet wird, kommt ein anderer Algorithmus als bei JSP-Objekten zur Anwendung: Formularinstanzen werden über eine anzugebene JSP-Seite statifiziert. Im Spezialattribut `generate_static` steht die OID der JSP-Seite, die die eigentliche Statifizierung übernimmt. Hat dieses Spezialattribut in einer Formularinstanz keinen Wert, wird ersatzweise in der Formularvorlage der Formularinstanz nachgesehen, ob ein solcher Wert existiert. Ist das der Fall, wird dieser Wert für die Statifizierung herangezogen. Der ermittelten JSP-Seite wird zur Statifizierung die OID der Formularinstanz übergeben. Das Ergebnis (im Allgemeinen eine HTML-Seite) wird anschließend vom Deploymentssystem erzeugt. Referenzen von anderen Objekten auf die Formularinstanz werden automatisch auf diese HTML-Seite umgesetzt.

Wenn für das Deploymentsystem die Option *Statifizierte Seiten analysieren* aktiviert ist, werden bei jeder Änderung der JSP-Seite sämtliche Formularinstanzen, denen diese JSP-Seite zugewiesen ist, neu erzeugt und statifiziert.

Die JSP-Seite bestimmt das Layout bzw. die Präsentation der Daten, die aus der Formularinstanz stammen. Ist keine OID im Spezialattribut `generate_static` hinterlegt, wird nur der unveränderte Inhalt der Formularinstanz in Form von XML-Code vom Deploymentsystem erzeugt.

Beim Anlegen eines Deploymentsystems wird im Admin-Client festgelegt, wie die Referenzen in den WCM-Objekten in URLs umgewandelt werden. Dabei gibt es folgende Optionen:

- *absolut*: In den Links werden absolute Pfade verwendet.

Beispiel: `<a href="http://www.company.example/products/NewProducts.htm">`

- *relativ*: In den Links werden relative Pfade verwendet.

Beispiel: `<a href="NewProducts.htm">`

- *Server-relativ*: In den Links werden relative Referenzen ohne Protokollangabe und Basis-URL verwendet.

Beispiel: `<a href="/products/NewProducts.htm">`

Mithilfe des Spezialattributs `absolute_urls` können Sie im Metadaten-Dialog des Content-Clients die Einstellung aus dem Admin-Client überschreiben.

Deploymentsysteme, die mit der Option *relativ* angelegt wurden, berechnen Referenzen, die die JSP-Seite enthält, relativ zum Pfad der JSP-Seite. Wird die JSP-Seite dann zwecks Statifizierung einer Formularinstanz ausgeführt, gelangen die relativen URLs in den statischen Code der Formularinstanz, wo sie nicht mehr angepasst werden. Dies ist dann problematisch, wenn sich die Formularinstanz nicht im gleichen Verzeichnis wie die für die Statifizierung verwendete JSP-Seite befindet. Die relativen URLs sind in diesem Fall falsch. Um dieses Problem zu umgehen, setzen Sie für die JSP-Seite das Spezialattribut `absolute_urls` auf den Wert `on` (oder auf jeden anderen Wert als `SERVER_RELATIVE`). Dadurch werden auf der ausgeführten JSP-Seite absolute URLs erzeugt. Diese URLs werden bei der Statifizierung der Formularinstanz dann in relative URLs umgewandelt, und zwar relativ zu der gerade erzeugten Formularinstanz.

Statifizierung von XML-Objekten

XML-Objekte können statifiziert werden. Bei der Statifizierung von XML-Objekten werden zwei Fälle unterschieden:

- Dem XML-Objekt ist eine Formularvorlage zugewiesen **oder** das Spezialattribut `generate_static` des XML-Objekts hat einen rein numerischen Wert.

Die Statifizierung erfolgt auf die gleiche Weise wie die Statifizierung von Formularinstanzen, d. h. mithilfe einer JSP-Seite (siehe Abschnitt "Statifizierung von Formularinstanzen" auf Seite 43). Hierbei werden für jedes XML-Objekt folgende Dateien erzeugt:

- die nicht statifizierte Seite
- die statifizierte Seite
- falls erforderlich: die Surrogatseite mit einem Link auf das XML-Dokument

Hinweis: Bei Nutzung einer Formularvorlage wird der Inhalt der nicht statifizierten Seite als HTML-Code (Dateiendung **.html**) verteilt, da die Formularvorlage berücksichtigt wird. Dies hat zur Folge, dass die JSP-Seite, die die Statifizierung übernimmt, nicht direkt den XML-Code des XML-Objekts auswerten kann. Daher ist es nicht empfehlenswert, einem XML-Objekt eine Formularvorlage zuzuordnen.

- Das Spezialattribut `generate_static` des XML-Objekts hat keinen numerischen Wert, dem XML-Objekt ist keine Formularvorlage zugewiesen. Mindestens eine Vorlage, die dem XML-Objekt zugeordnet ist, ist eine JSP-Vorlage. Die Vorlage, die dem XML-Objekt direkt zugeordnet ist, sollte in diesem Fall eine XSLT-Vorlage sein.

Die Statifizierung erfolgt auf die gleiche Weise wie die Statifizierung von JSP-Objekten (siehe Abschnitt “Statifizierung von JSP-Objekten” auf Seite 42). Dabei werden für jedes XML-Objekt folgende Dateien erzeugt:

- die statifizierte Seite
- falls erforderlich: die Surrogatseite mit einem Link auf das XML-Dokument

Staging

Alle Objekte einer WCM-verwalteten Website durchlaufen folgende Stufen: Bearbeitung (Editieren), Qualitätssicherung und Veröffentlichung im Produktionsbetrieb. Die Bearbeitung und Qualitätssicherung der Objekte erfolgt im Content-Client. Nach der Veröffentlichung erfolgt der Zugriff auf die Objekte über einen Browser. Das Durchlaufen dieser Stufen wird als *Staging* bezeichnet.

Durch die Methoden des Portal Manager API – insbesondere die Methoden des `VipObjectHandlerBean` – werden die Objekte einer WCM-verwalteten Website bearbeitet. Einige dieser Methoden ändern die Inhalte bzw. die Metadaten eines Objekts, andere führen zu einer Änderung des Objektstatus. Damit wird das mehrstufige Staging von Livelink WCM Server abgebildet. Dieses Staging entspricht dem Lebenszyklus eines WCM-Objekts, der im nachfolgenden Diagramm dargestellt ist.

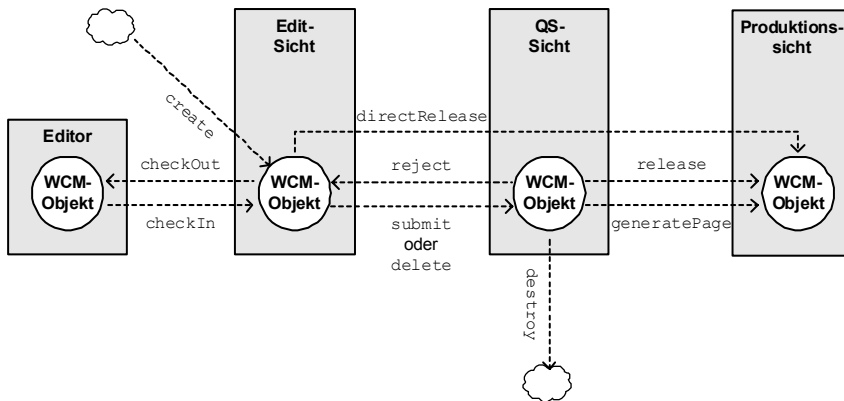


Abb. 5 – Der Lebenszyklus eines WCM-Objekts

Der Lebenszyklus eines WCM-Objekts kann folgende Übergänge enthalten, die durch die Methoden des `VipObjectHandlerBean` abgebildet werden:

- Das Objekt wird in der Edit-Sicht angelegt (`create`).
- Um den Inhalt des Objekts zu pflegen, kann es in der Edit-Sicht ausgeliehen und in einem Editor bearbeitet werden (`checkOut`).
- Nach dem Ändern muss das Objekt in der Edit-Sicht wieder zurückgegeben werden (`checkIn`).
- In die QS-Sicht gelangt das Objekt dadurch, dass es in der Edit-Sicht zur Qualitätssicherung vorgelegt wird (`submit`).
- In der QS-Sicht kann das Objekt freigegeben und damit in die Produktionssicht übertragen werden (`release`).
- In der Edit-Sicht kann das Objekt direkt freigegeben werden, wenn für das Objekt das Metadatum "Direkte Freigabe" gesetzt ist und das Objekt mit dieser Einstellung bereits einmal freigegeben wurde (`directRelease`).

- Das Objekt kann in der QS-Sicht aber auch abgelehnt und an die Edit-Sicht zurückgeschickt werden (`reject`).
- Das Neuerzeugen eines bereits freigegebenen dynamischen Objekts, das in der Produktionssicht statifiziert wurde, führt dazu, dass dieses Objekt in der Produktionssicht aktualisiert wird (`generatePage`).
- Das Löschen eines Objekts erfolgt in der Edit-Sicht (`delete`). Wenn es sich um ein bereits freigegebenes Objekt handelt, muss das Objekt anschließend in der QS-Sicht vernichtet werden (`destroy`). Wird in der QS-Sicht das Löschen allerdings abgelehnt (`reject`), ist das Objekt auch in der Edit-Sicht wieder vorhanden.

Weitere Angaben zum Staging finden Sie im Content-Client-Benutzerhandbuch. Weitere Informationen zum `VipObjectHandlerBean` finden Sie im Abschnitt “Die Klasse `VipObjectHandlerBean`” auf Seite 128. Eine detaillierte Beschreibung der Methoden dieser Klasse liefert die Javadoc zum Portal Manager API.

2.4 Integration externer Anwendungen

Das Portal Manager API ist im Wesentlichen ein Framework, das auch die Integration externer Anwendungen, wie etwa E-Commerce-Anwendungen, zulässt. Dabei spielen Themen wie Dynamisierung, Personalisierung und Korrelation mit Website-Content eine herausragende Rolle.

Die Integration einer externen Anwendung ist relativ einfach und im Portal Manager API sehr gut vorbereitet. Wenn Sie externe Anwendungen im Rahmen des gegebenen Frameworks implementieren möchten, sollten Sie einige Designgrundsätze befolgen. Diese lassen sich am leichtesten durch eine Bottom-up-Strategie umsetzen:

- In den meisten Fällen spielt eine externe Anwendung ausschließlich die Rolle einer Datenquelle und -senke. Hierzu müssen Sie entsprechende Repositories implementieren oder bereits existierende Repository-Implementationen heranziehen. Repositories sind wichtige (im Allgemeinen die wichtigsten oder gar die einzigen) Bestandteile einer Application.

Um neue Repository-Implementationen zu erstellen, können Sie die schon existierenden Repositories erweitern oder auf Grundlage des abstrakten Basis-Repository (`RepositoryImpl`) neue entwickeln.

- Für jede neue Anwendung ist die Entwicklung einer Application-Klasse nötig. Diese beschreibt die grundlegenden Funktionen und das grundlegende Verhalten der Anwendung. Auf dieser Ebene sollten jedoch keine Funktionen zu Funktionseinheiten zusammengefasst werden.

Sie erhalten eine neue Application-Klasse sehr schnell, wenn Sie von der mitgelieferten abstrakten Basisimplementation (`ApplicationImpl`) ableiten.

Ein wesentlicher Bestandteil jeder Application ist ein Repository (oder mehrere Repositories). Für die meisten Anwendungen müssen Sie darüber hinaus gar keine weiteren Funktionen entwickeln, da die Anwendung durch das Repository schon ausreichend beschrieben ist. Beispielsweise haben die Applications, die verschiedene Aspekte von Livelink WCM Server implementieren, keine weiteren Methoden als die der abstrakten Basisimplementation für Applications.

- Jede Application wird im Bedarfsfall von einem Bean oder von mehreren Beans gekapselt. Dieses Bean müssen Sie ebenfalls neu entwickeln. Auch dafür ist es empfehlenswert, auf einer vorgegebenen abstrakten Basisimplementation aufzusetzen.

Das Bean übernimmt zwei Aufgaben: Zunächst bildet es die mehr oder weniger formale Schnittstelle zum JSP-Skript und zum Session-Handling der JSP-Engine. Diese Aufgabe wird von der Basisimplementation des Bean übernommen. Die weitere, anspruchsvollere Aufgabe des Bean ist, einfache und mächtige Zugriffsmethoden auf möglichst hohem Abstraktionsniveau für eine Application und ihre Repositories zur Verfügung zu stellen. In dem Bean sind die von einer Application zur Verfügung gestellten Basisfunktionalitäten zusammen mit den Zugriffsmöglichkeiten der Repositories zu komplexeren Methoden zusammengefasst, die in sich konsistent sind. Diese Methoden sollten darüber hinaus einfach sein und die Möglichkeiten der zugrunde liegenden Anwendung möglichst "natürlich" widerspiegeln.

Letztendlich sollten Sie in einem JSP-Skript möglichst nur die Methoden des Bean benutzen, obwohl natürlich auch immer der Zugriff auf die Application und sogar auf die Repositories möglich ist.

Die Basisimplementation enthält Vorkehrungen, mit denen Sie die Benutzeroberfläche in unterschiedlichen Sprachen präsentieren können.

- Um eine externe Anwendung über den Browser steuern zu können, müssen Sie eine HTML-Seite mit einem JSP-Skript entwickeln. Die HTML-Seite muss das zuvor entstandene Bean nutzen und ein WCM-verwaltetes Objekt sein. Es ist also erforderlich, dass Sie die HTML-Seite in die Website integrieren, über die die zugrunde liegende externe Anwendung erreicht werden soll.

Wenn Sie die genannten Designvorgaben einhalten, ergibt sich automatisch die für das Portal Manager API typische Architektur, die bereits aufgezeigt wurde. Die folgende Abbildung stellt den relevanten Ausschnitt leicht modifiziert dar:

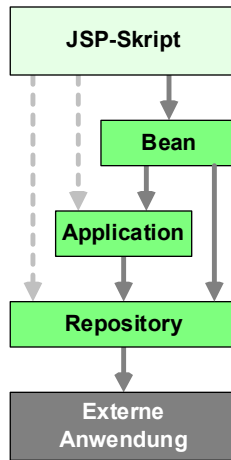


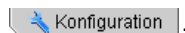
Abb. 6 – Prinzipielles Klassendesign im Portal Manager API

KAPITEL 3

Portal Manager API verwalten

Dieses Kapitel unterstützt Sie bei der Konfiguration des Portal Manager API. Es enthält außerdem Hinweise zum Einsatz eines LDAP-Verzeichnisdienstes und zur Fehlersuche.

Die gesamte Konfiguration Ihres WCM-Systems erfolgt über den Admin-Client. An dieser zentralen Stelle nehmen Sie also auch die Administration und Konfiguration des Portal Manager API vor. Um die Konfiguration zu öffnen, starten Sie den Admin-Client und wählen das Register



In einer baumartigen Ansicht sehen Sie die Elemente, die Sie in dieser Ansicht verwalten können.

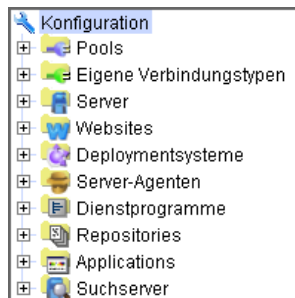


Abb. 7 – Die Elemente der Konfiguration

In diesem Kapitel werden die folgenden Elemente der Konfiguration beschrieben:

Tabelle 2 – Funktionen für die Konfiguration von Repositories und Applications

Element	Verfügbare Funktionen
<i>Repositories</i>	Erstellen, Bearbeiten und Löschen von Repositories Siehe Abschnitt 3.1 "Repositories verwalten" ab Seite 55
<i>Applications</i>	Erstellen, Bearbeiten und Löschen von Applications Siehe Abschnitt 3.2 "Applications verwalten" ab Seite 69

Die Verwaltung der übrigen Elemente und die Bedienung des Admin-Clients werden ausführlich im Livelink WCM Server-Administratorhandbuch beschrieben.

3.1 Repositories verwalten

Allgemeine Informationen zu Repositories erhalten Sie im Abschnitt “Repositories” auf Seite 111.

Wenn Sie das Element *Repositories* öffnen, werden Ihnen alle verfügbaren Repositories angezeigt – sowohl im Baum links als auch in einer Liste im rechten Fensterbereich.

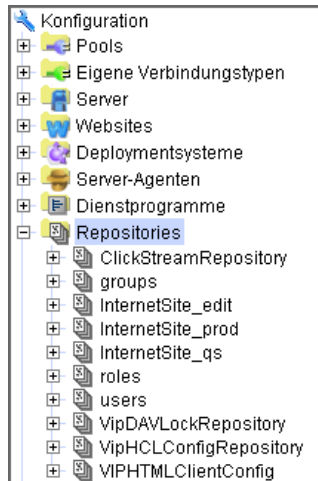


Abb. 8 – Übersicht über die verfügbaren Repositories

Die hier angezeigten Repositories ClickStreamRepository, groups, roles, users, VipDAVLockRepository, VipHCLConfigRepository und VIPHTMLClientConfig werden während der Installation des WCM-Systems angelegt. Diese Repositories gehören zum Lieferumfang des Portal Manager API und sollten nicht gelöscht werden. Die Repositories InternetSite_edit, InternetSite_prod und InternetSite_qs wurden automatisch beim Einrichten der Website “InternetSite” hinzugefügt.

Folgende Funktionen stehen für Repositories zur Verfügung:

- neues Repository einrichten (siehe folgenden Abschnitt)
- Repositories über Parameter konfigurieren (siehe Abschnitt “Parameter eines Repository festlegen” auf Seite 58)
- Einstellungen von Repositories bearbeiten (siehe Abschnitt “Allgemeine Einstellungen von Repositories” auf Seite 65)
- Übersicht über zugeordnete Applications, Zuordnen neuer Applications und Aufheben von Zuordnungen (siehe Abschnitt “Application zu einem Repository zuordnen” auf Seite 65)
- Repository entfernen (siehe Abschnitt “Repository löschen” auf Seite 68)

Repository anlegen

So legen Sie ein Repository an:

1. Wählen Sie *Konfiguration* → *Repositories*.
2. Wählen Sie *Neues Repository* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Anlegen eines Repository

Der Dialog zum Anlegen eines Repository wird geöffnet.

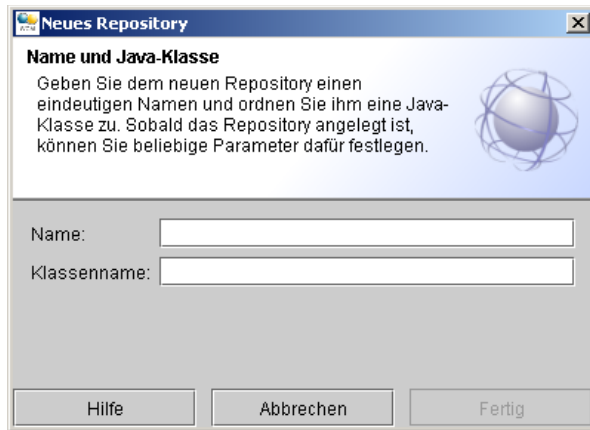


Abb. 9 – Repository anlegen

3. Geben Sie im Feld *Name* einen Namen für das neue Repository an. Dieser Name ist frei wählbar, muss jedoch innerhalb des WCM-Systems eindeutig sein.

Für Repositories vom Typ `VipObjectRepository` und `VipObjectHandlerRepository` kann dies der Name des Deployment-Systems sein.

Der Name des Repository kann nachträglich nicht geändert werden.

4. Geben Sie im Feld *Klassenname* den Namen der Java-Klasse des Repository an.

Die Klasse wird vom Class-Loader über diesen Namen geladen. Es wird erwartet, dass sie das Interface `Repository` implementiert und einen öffentlichen leeren Konstruktor hat.

5. Klicken Sie auf die Schaltfläche *Fertig*.

Parameter eines Repository festlegen

Die Konfiguration von Repositories erfolgt über Parameter. Art und Anzahl der Parameter hängen von den Funktionen des Repository ab.

Parameter können als Knoten oder einzelne Parameter definiert werden. In Knoten werden Parameter desselben Typs zusammengefasst. Knoten können wiederum weitere Knoten enthalten. Im Unterschied zu Parametern haben sie keinen Wert.

So definieren Sie Parameter für ein Repository:

1. Wählen Sie *Konfiguration* → *Repositories* → {*Repository-Name*} → *Parameter*.
2. Wählen Sie *Neuer Parameter* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Hinzufügen eines Parameters

Der Dialog *Neuer Knoten* wird geöffnet.



Abb. 10 – Anlegen eines Parameters für ein Repository

3. Wählen Sie hier, ob Sie einen *Knoten* oder einen *Parameter* anlegen möchten.

Wenn Sie dem Repository viele Parameter zuordnen möchten, ist es sinnvoll, die Parameter zu gruppieren. Dazu legen Sie zunächst Knoten an und dann unter den Knoten jeweils einen oder mehrere Parameter.

4. Wenn Sie die Option *Knoten* gewählt haben, geben Sie einen eindeutigen Namen für den neuen Knoten ein.

Haben Sie die Option *Parameter* gewählt, geben Sie einen Namen für den Parameter und einen Wert ein.

5. Klicken Sie auf die Schaltfläche *OK*.

Hinweis: Die Repositories `ClickStreamRepository` und `VipDAVLockRepository` speichern ihre Daten in einer Datenbanktabelle. Sie können nur genutzt werden, wenn ihnen ein JDBC-Pool zugeordnet ist. In diesem JDBC-Pool werden die Verbindungen zur Datenbank verwaltet. Um einen JDBC-Pool zuzuordnen, geben Sie den Namen des entsprechenden Pools als Wert für den Parameter `store/{Tabellenname}/poolname/default` an.

Die folgende Tabelle liefert Ihnen eine Übersicht über die Parameter, die die mit dem Portal Manager API ausgelieferten Repositories erwarten.

Tabelle 3 – Repositories und ihre Parameter

Repository	
Parameter	Beschreibung
VipHclConfigRepository	
ProfileDataRefresh-Milliseconds	Die Zeit (in Millisekunden), die ein ungeändertes Profil im Content-Client vorgehalten wird, bevor es erneut vom Administrationsserver geladen wird
SaveWaitMilli-seconds	Der Abstand (in Millisekunden) in dem Änderungen an den Profildaten gespeichert werden
DirectoryRepository	
authentication	Die Authentifizierungsmethode. Es sind die Werte SIMPLE (keine Verschlüsselung) und NONE möglich. NONE sollte nicht benutzt werden, weil in diesem Fall keine Autorisierung durchgeführt wird und damit keine Daten gelesen und geschrieben werden können.
base	Der Startknoten, unter dem die Einträge gesucht oder eingefügt werden Beispiel: ou=software, o=company.example
build-profile_filter_expr	Der Suchausdruck, der bei der Suche nach Gruppen und Rollen benutzt wird. Der Ausdruck *DN* wird dabei durch den "distinguished name" ersetzt. Der Ausdruck *UID* wird ersetzt durch die User-ID, wenn der Methode keine User-ID übergeben wurde, durch den "distinguished name".
context-factory	Die "Initial Context Factory" für die LDAP-Klassen. Zurzeit wird nur die Klasse com.sun.jndi.ldap.LdapCtxFactory unterstützt.
credentials	Das Passwort des unter principal angegebenen Benutzers

Repository	
Parameter	Beschreibung
credentials-changes	Das Passwort des unter principal-changes angegebenen Benutzers
credentials-read	Das Passwort des unter principal-read angegebenen Benutzers
dn-pattern	Das Pattern für den Aufbau des "distinguished name" (dn). * wird durch den Schlüsselwert ersetzt. Beispiel: uid=*, ou=software, o=company.example wird zu uid=smith, ou=software, o=company.example, wenn der aktuelle Schlüsselwert smith ist.
group-base	Der Startknoten, unter dem die Einträge für die Gruppen des Benutzers gesucht werden. Wenn das Attribut nicht gesetzt ist, wird der Wert von base für dieses Attribut genommen.
mappings	Das Mapping zwischen den LDAP-Parametern und den WCM-Parametern. Mögliche Mappings sind z.B. name und language. Diese haben die Attribute ldapattr und pomaattr. Das Attribut pomaattr darf nicht geändert werden (vgl. Abbildung 22 "Das DirectoryRepository mit seinen Parametern" auf Seite 93).
member-key-attr	Der Name des LDAP-Attributes, unter dem die Benutzer einer Gruppe bzw. Rolle gespeichert werden.
objectclass	Der Name der Objektklasse, die beim Anlegen eines neuen Eintrags eingefügt werden soll, sofern keine Objektklasse spezifiziert ist (Beispiel: vipUser). Die Einträge sind durch Leerzeichen separiert.
password-attr	Der Name des LDAP-Attributs, das in Livelink WCM Server als Passwort verwendet wird

Repository	
Parameter	Beschreibung
principal	Ein LDAP-Benutzer mit dem Recht, Einträge hinzuzufügen Beispiel: uid=admin, ou=software, o=company.example
principal-changes	Ein LDAP-Benutzer mit dem Recht, schreibend auf Einträge im LDAP-Verzeichnisdienst zuzugreifen. Wenn dieser Parameter nicht gesetzt ist, wird das Ändern und Löschen von Einträgen im Kontext des angemeldeten Benutzers durchgeführt.
principal-read	Ein LDAP-Benutzer mit dem Recht, lesend auf Einträge im LDAP-Verzeichnisdienst zuzugreifen. Wenn dieser Parameter nicht gesetzt ist, wird der lesende Zugriff im Kontext des angemeldeten Benutzers durchgeführt.
role-base	Der Startknoten, unter dem die Einträge für die Rollen des Benutzers gesucht werden. Wenn das Attribut nicht gesetzt ist, wird der Wert von base für dieses Attribut genommen.
search-scope	Beschreibt den Suchbereich unterhalb des in dn-pattern angegebenen Startknotens. SUBTREE_SCOPE sucht im ganzen Unterbaum ab BASE, ONELEVEL_SCOPE sucht nur in der Ebene direkt unter BASE.
url	Die durch Leerzeichen separierte Liste von URLs zu Directory-Servern. Zunächst wird der erste LDAP-Server angesprochen. Schlägt dies fehl, wird versucht, den zweiten anzusprechen usw. Beispiel: ldap://ldap.company.example:389 ldap://ldap2.company.example:389

Repository	
Parameter	Beschreibung
DoorwaysRepository	
alias	Ein Alias für das DoorwaysRepository. Wenn kein Alias spezifiziert ist, wird der Name des DoorwaysRepository verwendet, um die Verbindung zum Doorways-Server aufzubauen. Dieser Parameter ist optional.
authentication	Die Authentifizierungsmethode. Es sind die Werte LOGIN, SESSION und CONFIG möglich. LOGIN: Für die Klasse DoorwaysBean muss über die Methode DoorwaysBean.setLogin eine explizite Anmeldung erfolgen. SESSION: Es werden die Anmeldedaten aus der Klasse SessionBean verwendet. CONFIG: Alle Benutzer werden mit den Daten eines Standardbenutzers angemeldet. Diese Daten werden in den Parametern user und pwd spezifiziert.
hostname	Der Hostname des Doorways-Servers, zu dem eine Verbindung hergestellt werden soll
port	Der Port des Doorways-Servers für die Verbindung
pwd	Das Passwort des unter user angegebenen Standardbenutzers
user	Der Standardbenutzer, dessen Daten für die Authentifizierungsmethode CONFIG verwendet wird

Repository	
Parameter	Beschreibung
VipObjectHandlerRepository (für Deploymentsysteme)	
deployment-name	Der Name des Deploymentsystems, z.B. {Website-Name}_edit, auf dessen Objekte über das Repository zugegriffen wird Hinweise: Alternativ kann das Repository auch durch Angabe der Parameter <code>view</code> und <code>website</code> konfiguriert werden. Auf diese Weise kann auf die Objekte einer Website zugegriffen werden, ohne Deploymentsysteme einzurichten. Repositories, die so konfiguriert sind, können keine Deployment-spezifischen Objektdaten (z. B. URL, Pfad) zurückliefern. Der Parameter <code>deployment-name</code> hat Vorrang vor den beiden Parametern <code>view</code> und <code>website</code>, d.h., ist der Parameter <code>deployment-name</code> spezifiziert, werden die eventuell ebenfalls spezifizierten Parameter <code>view</code> und <code>website</code> ignoriert.
metadata-style	Das verwendete Metadatenschema. In diesem Parameter wird festgelegt, ob das Repository die Attributnamen und -werte entsprechend der Version 5e oder 8 verwendet. Mögliche Werte sind 5e und 8. Eine vergleichende Übersicht hierzu finden Sie im Anhang A "Metadatenschemata".

Parameter eines Repository löschen

Um einen Parameter aus der Konfiguration eines Repository zu entfernen, markieren Sie den zu löschenden Parameter im rechten Fensterbereich. Wählen Sie *Parameter löschen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Löschen eines Parameters

Allgemeine Einstellungen von Repositories

So bearbeiten Sie die Einstellungen eines Repository:

1. Wählen Sie *Konfiguration* → *Repositories*.
2. Markieren Sie das gewünschte Repository im linken Baum.

Die Einstellungen werden im rechten Fensterbereich angezeigt. Sie können den Namen der Java-Klasse des Repository nachträglich ändern. Der Name des Repository selbst wird während des Anlegens festgelegt und kann später nicht geändert werden.

3. Klicken Sie auf die Schaltfläche *Übernehmen*.

Application zu einem Repository zuordnen

Über *Konfiguration* → *Repositories* → *{Repository-Name}* → *Applications* erhalten Sie eine Übersicht über die Applications, die diesem Repository zugeordnet sind. Die zugeordneten Applications werden im rechten Fensterbereich in einer Liste angezeigt.

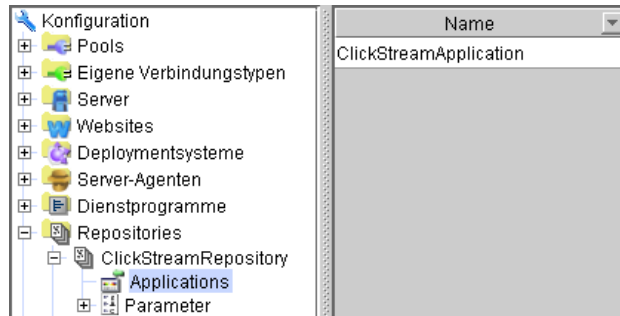


Abb. 11 – Zu einem Repository zugeordnete Applications

Zuordnung einer Application zu einem Repository herstellen

So ordnen Sie eine Application zu einem Repository zu:

1. Wählen Sie *Konfiguration* → *Repositories* → {*Repository-Name*} → *Applications*.
2. Wählen Sie *Application zuordnen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Zuordnen einer Application zu einem Repository

Der Dialog *Application auswählen* wird geöffnet. Er zeigt eine Liste der noch nicht zugeordneten Applications.

3. Markieren Sie die gewünschte(n) Application(s).
4. Klicken Sie auf die Schaltfläche *OK*.

Zuordnung einer Application zu einem Repository aufheben

Um die Zuordnung einer Application zu einem Repository wieder aufzuheben, markieren Sie die entsprechende Application im rechten Fensterbereich. Wählen Sie *Zuordnung der Application aufheben* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Aufheben der Zuordnung einer Application zu einem Repository

Struktur unterhalb des Elements *Repositories*

Jedes angelegte Repository hat die Knoten *Applications* und *Parameter*. Wenn Sie dem Repository Applications oder Parameter zugeordnet haben, erscheinen diese Einträge unterhalb des jeweiligen Knotens (in der folgenden Abbildung wurden dem ClickStreamRepository die Parameter max-records, store, table und tosql zugeordnet).

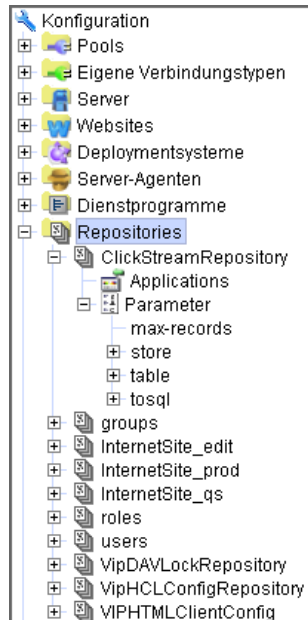


Abb. 12 – Struktur unterhalb des Elements *Repositories*

Repository löschen

So löschen Sie ein Repository:

1. Wählen Sie *Konfiguration* → *Repositories*.
2. Markieren Sie das gewünschte Repository im linken Baum.
3. Wählen Sie *Repository löschen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Löschen eines Repository

4. Bestätigen Sie die Sicherheitsabfrage durch Klicken auf die Schaltfläche *Ja*.

Die Konfiguration des Repository wird aus dem WCM-System entfernt, die dazugehörige Java-Klasse wird jedoch nicht gelöscht.

3.2 Applications verwalten

Allgemeine Informationen zu Applications erhalten Sie im Abschnitt “Applications” auf Seite 116.

Wenn Sie das Element *Applications* öffnen, werden Ihnen alle verfügbaren Applications angezeigt – sowohl im Baum links als auch in einer Liste im rechten Fensterbereich.

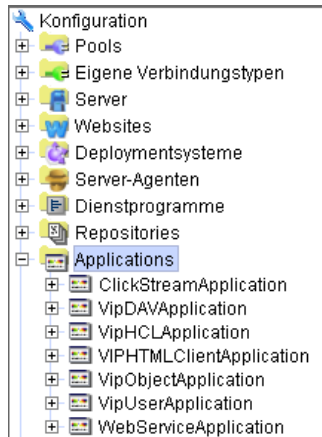


Abb. 13 – Übersicht über die verfügbaren Applications

Die hier angezeigten Applications ClickStreamApplication, VipDAVApplication, VipHCLApplication, VIPHTMLClientApplication, VipObjectApplication, VipUserApplication und WebServiceApplication werden während der Installation des WCM-Systems angelegt. Diese Applications gehören zum Lieferumfang des Portal Manager API und sollten nicht verändert oder gelöscht werden. Sie können hier jedoch beliebige eigene Applications hinzufügen und bearbeiten.

Folgende Funktionen stehen für Applications zur Verfügung:

- neue Application einrichten (siehe folgenden Abschnitt)

- Applications über Parameter konfigurieren (siehe Abschnitt “Parameter einer Application festlegen” auf Seite 75)
- Einstellungen von Applications bearbeiten (siehe Abschnitt “Allgemeine Einstellungen von Applications” auf Seite 85)
- Übersicht über zugeordnete Repositories, Zuordnen neuer Repositories und Aufheben von Zuordnungen (siehe Abschnitt “Repository zu einer Application zuordnen” auf Seite 86)
- Application löschen (siehe Abschnitt “Application löschen” auf Seite 88)
- Application zu einem Content-Server zuordnen (siehe Abschnitt “Application zu einem Content-Server zuordnen” auf Seite 89)

Application anlegen

So legen Sie eine Application an:

1. Wählen Sie *Konfiguration* → *Applications*.
2. Wählen Sie *Neue Application* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Anlegen einer Application

3. Folgen Sie den Anweisungen des Assistenten.

Basisdaten festlegen

Im ersten Dialog legen Sie die Basisdaten der Application fest.

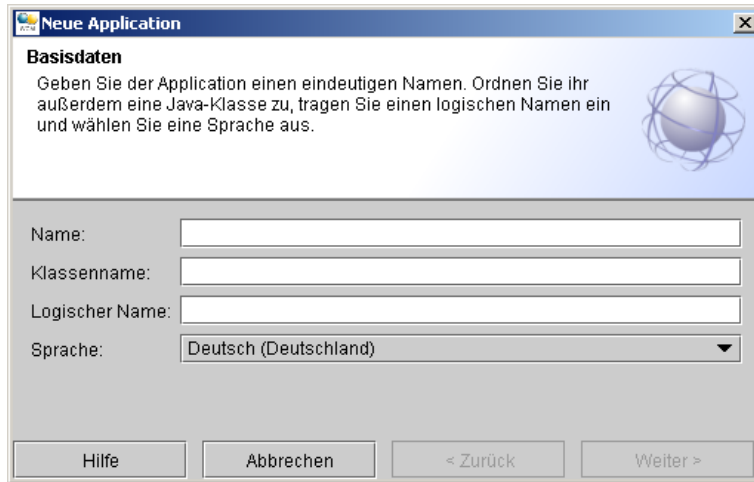


Abb. 14 – Basisdaten der Application festlegen

- **Name:** Der Name der Application muss eindeutig sein und mit einem Buchstaben beginnen.

Der Name der Application kann nachträglich nicht geändert werden.

- **Klassenname:** Name der Java-Klasse der Application. Die Klasse wird vom Class-Loader über diesen Namen geladen. Es wird erwartet, dass sie das Interface `Application` implementiert und einen öffentlichen leeren Konstruktor hat.
- **Logischer Name:** Der logische Name der Application ist der Name, unter dem die Application vom Portal Manager API angesprochen wird. Sollen z. B. zwei Content-Server unterschiedlich konfigurierte `VipUserApplications` besitzen, so können zwei `VipUserApplications` mit unterschiedlichen Namen erzeugt

werden, die beide den logischen Namen “VipUserApplication” haben.

- **Sprache:** Standardsprache der Application. Die Sprache wird u.a. dazu verwendet, die Ressourcen lokalisiert auszulesen. Es wird eine Angabe vom Typ {Sprachcode}_{Ländercode}, z.B. en_US, erwartet. Weitere Informationen hierzu finden Sie in der Java SDK-API-Dokumentation der Klasse `java.util.Locale`.

Bestätigen Sie die Angaben durch Klicken auf die Schaltfläche *Weiter*.

Assistent beenden?



Abb. 15 – Assistent beenden?

Die Angaben, die Sie bis jetzt gemacht haben, genügen zum Anlegen der neuen Application. Sie können der Application jedoch optional Repositories und Ressourcen zuordnen.

Aktivieren Sie in diesem Dialog das gewünschte Optionsfeld, und bestätigen Sie die Auswahl durch Klicken auf die Schaltfläche *Fertig* bzw. *Weiter*.

Repositories zuordnen

In diesem Dialog können Sie der Application bereits beim Anlegen Repositories zuordnen.



Abb. 16 – Repository beim Anlegen einer Application zuordnen

Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie auf die Schaltfläche *Hinzufügen*.
2. Markieren Sie in der eingeblendeten Liste die Repositories, die Sie zuordnen möchten
3. Bestätigen Sie die Auswahl durch Klicken auf die Schaltfläche *OK*.
Die Repositories werden in den Assistenten übernommen.
4. Klicken Sie auf die Schaltfläche *Weiter*.

Ressourcen zuordnen

Die Ressourcen bilden die Grundlage für die Internationalisierung in den Beans. In diesem Dialog können Sie den Zugriff auf Java-Ressourcen bereits beim Anlegen der Application einstellen.

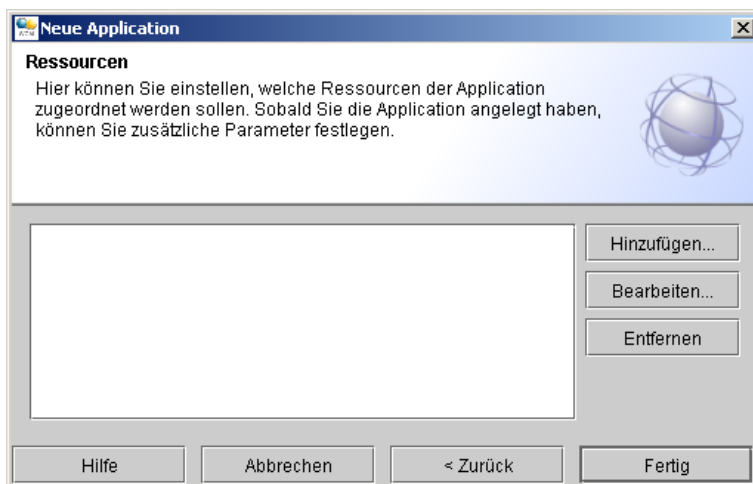


Abb. 17 – Ressourcen beim Anlegen einer Application zuordnen

Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie auf die Schaltfläche *Hinzufügen*.
2. Geben Sie in dem eingeblendeten Dialog den Ressourcen-Namen ein.

Die hier angegebenen Ressourcen werden über den Klassenpfad geladen und müssen den Java-Ressourcen entsprechen (weitere Informationen finden Sie in der Java SDK-API-Dokumentation der Klasse `java.util.ResourceBundle`).

3. Bestätigen Sie die Angaben durch Klicken auf die Schaltfläche *OK*.
Die Ressource wird in den Assistenten übernommen.

4. Klicken Sie auf die Schaltfläche *Fertig*.

Parameter einer Application festlegen

Die Konfiguration von Applications erfolgt über Parameter. Art und Anzahl der Parameter hängen von den Funktionen der Application ab.

Parameter können als Knoten oder einzelne Parameter definiert werden. In Knoten werden Parameter desselben Typs zusammengefasst. Knoten können wiederum weitere Knoten enthalten. Im Unterschied zu Parametern haben sie keinen Wert.

Hinweis: Die Applications, die zum Lieferumfang des Portal Manager API gehören (ClickStreamApplication, VipDAVApplication, VipHCLApplication, VIPHTMLClientApplication, VipObjectApplication, VipUserApplication und WebServiceApplication) sollten Sie nicht löschen. Auch die Basisdaten dieser Applications sollten nicht geändert werden.

So definieren Sie Parameter für eine Application:

1. Wählen Sie *Konfiguration* → *Applications* → *{Application-Name}* → *Parameter*.
2. Wählen Sie *Neuer Parameter* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Hinzufügen eines Parameters

Der Dialog *Neuer Knoten* wird geöffnet.

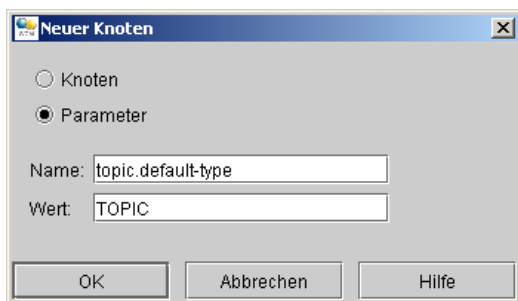


Abb. 18 – Anlegen eines Parameters für eine Application

3. Wählen Sie hier, ob Sie einen *Knoten* oder einen *Parameter* anlegen möchten.

Wenn Sie der Application viele Parameter zuordnen möchten, ist es sinnvoll, die Parameter zu gruppieren. Dazu legen Sie zunächst Knoten an und dann unter den Knoten jeweils einen oder mehrere Parameter.

4. Wenn Sie die Option *Knoten* gewählt haben, geben Sie einen eindeutigen Namen für den Knoten an.

Haben Sie die Option *Parameter* gewählt, geben Sie einen Namen für den Parameter und einen Wert an.

5. Klicken Sie auf die Schaltfläche *OK*.

Die folgende Tabelle liefert Ihnen eine Übersicht über die Parameter, die die mit dem Portal Manager API ausgelieferten Applications erwarten.

Tabelle 4 – WCM-Applications und ihre Parameter

Application	
Parameter	Beschreibung
ClickStreamApplication	
keine	keine
VipDAVApplication	
import-indexfile-name	Gibt den Namen von Indexdateien in Verzeichnissen an. Wenn neue Ressourcen erzeugt werden, werden Dateien mit diesem Dateinamen – unabhängig von der Dateiendung – als Indexdatei interpretiert. Der Inhalt dieser Dateien wird in die Indexdatei der entsprechenden Collection übernommen. Standardwert: index
lock-cleanup-interval	Gibt das Intervall (in Millisekunden) an, in dem nach abgelaufenen Sperrungen gesucht wird Standardwert: 600000 (entspricht 10 Minuten)
lock-infinite-timeout	Gibt die maximale Ablaufzeit einer Sperrung an Standardwert: 86400000 (entspricht 24 Stunden)

Application	
Parameter	Beschreibung
topic-default-type	WebDAV erlaubt das Erzeugen von Collections (Verzeichnisse, WCM-Objekte eines bestimmten Thementyps). Dieser Parameter gibt an, welchen Thementyp Objekte haben, die durch das WebDAV-Kommando MKCOL erzeugt werden. Standardwert: TOPIC
vip-default-attributes	Gibt den Namen der WCM-Metadaten an, die über die WebDAV-Schnittstelle abgefragt werden können Standardwert: oid, title, subtitle, description, target_group, keywords
VipHCLApplication	
default-profile-user	Gibt die Kennung des WCM-Benutzers an, dessen Standardprofil als Vorlage für die Standardprofile neuer Benutzer verwendet wird. Standardwert: leer (d.h., es gibt keine Vorlage für Standardprofile)
embedded-type-names	Enthält eine durch Kommas getrennte Liste von Objekttypen. Beim Anlegen eines WCM-Objekts werden Objekte dieses Typs als eingebettete Objekte betrachtet. Ist dieser Parameter nicht gesetzt, wird die folgende Liste als Standardwert verwendet: ETC, GIF, JAVASCRIPT, JPG, PNG

Application	
Parameter	Beschreibung
hcl-absolute-servlet	Gibt die absolute URL an, unter der das Controller-Servlet des Content-Clients angesprochen wird. Das Controller-Servlet ist für die Steuerung des Content-Clients zuständig. Beispiel: <code>http://wcmserver.company.example/hclservlet</code>
hcl-root	Gibt die absolute URL an, unter der die Startseite des Content-Clients angesprochen wird Beispiel: <code>http://wcmserver.company.example/cmsclient</code>
hcl-servlet-url	Gibt an, unter welchem Namen das Controller-Servlet des Content-Clients angesprochen werden soll. Das Controller-Servlet ist für die Steuerung des Content-Clients zuständig. Standardwert: <code>hclservlet</code>

Application	
Parameter	Beschreibung
{Name eines InSite Editing-Deploymentsystems}	Konfiguriert für das entsprechende InSite Editing-Deploymentsystem das Fenster mit den Objektinformationen. Folgende Parameter können unterhalb dieses Parameterknotens angelegt werden: keys: Durch Leerzeichen getrennte Liste von Attributen, die für jedes Objekt angezeigt werden. Standardmäßig werden oid, title, type, state und version angezeigt. width: Breite des Fensters in Pixeln height: Höhe des Fensters in Pixeln
VIPHTMLClientApplication	
DEFAULT_VALUE_FOR_APPLET_WORK_DIRECTORY	Gibt das lokale Arbeitsverzeichnis für das Download-Applet des Content-Clients (Classic) an. Der Wert darf nicht leer sein und muss ein gültiges Verzeichnis sein. Standardwert: c:\temp
DEFAULT_VALUE_FOR_AUTO_REFRESH_TIME	Gibt die Zeitspanne in Sekunden an, nach der die Aktionsliste des Content-Clients (Classic) automatisch aktualisiert wird. Der Wert muss eine ganze positive Zahl oder 0 sein. Standardwert: 0

Application	
Parameter	Beschreibung
DEFAULT_VALUE_FOR_NAVIGATION_APPLET_OBJECTLIST_POS	Gibt an, wo im Content-Client (Classic) bei Verwendung des Navigations-Applets die Objektliste angezeigt wird. Mögliche Werte sind: 0 = unterhalb der Themenstruktur 1 = rechts von der Themenstruktur 2 = innerhalb der Themenstruktur Standardwert: 0
DEFAULT_VALUE_FOR_NO_AUTO_TASKMANAGER_WINDOW	Gibt an, ob die Aktionsliste des Content-Clients (Classic) beim Start einer asynchronen Aktion automatisch geöffnet werden soll. Mögliche Werte sind <code>false</code> (öffnen) oder <code>true</code> (nicht öffnen). Standardwert: <code>false</code>
DEFAULT_VALUE_FOR_NO_OBJECT_PREVIEW_MODE	Gibt an, ob die Objektvorschau des Content-Clients (Classic) ausgeschaltet werden soll. Mögliche Werte sind <code>false</code> (nicht ausschalten) oder <code>true</code> (ausschalten). Standardwert: <code>false</code>
DEFAULT_VALUE_FOR_USE_DOWNLOAD_APPLET	Gibt an, ob das Download-Applet des Content-Clients (Classic) verwendet werden soll. Mögliche Werte sind <code>false</code> (nicht verwenden) und <code>true</code> (verwenden). Standardwert: <code>true</code>

Application	
Parameter	Beschreibung
DEFAULT_VALUE_FOR_USE_INTEGRATED_HTML_EDITOR	Gibt an, ob der integrierte HTML-Editor des Content-Clients (Classic) verwendet werden soll. Mögliche Werte sind <code>false</code> (nicht verwenden) und <code>true</code> (verwenden). Standardwert: <code>false</code>
DEFAULT_VALUE_FOR_USE_NAVIGATION_APPLET	Gibt an, ob das Navigations-Applet des Content-Clients (Classic) verwendet werden soll. Mögliche Werte sind <code>false</code> (nicht verwenden) und <code>true</code> (verwenden). Standardwert: <code>false</code>
DEFAULT_VALUE_FOR_USE_SEPARATE_PREVIEW_WINDOW	Gibt an, ob für die Objektvorschau des Content-Clients (Classic) ein separates Browserfenster geöffnet werden soll. Mögliche Werte sind <code>false</code> (kein separates Fenster) und <code>true</code> (separates Fenster). Standardwert: <code>false</code>
MAX_SHOWN_FILTER_RESULTS	Gibt an, wie viele Objekte nach dem Anwenden des Objektfilters maximal in der Ergebnisliste des Content-Clients (Classic) angezeigt werden Standardwert: 1000
MAX_SHOWN_LOG_ENTRIES	Gibt an, wie viele Einträge im Protokoll des Content-Clients (Classic) maximal angezeigt werden Standardwert: 100

Application	
Parameter	Beschreibung
MAX_SHOWN_PRINCIPALS	Gibt an, wie viele Einträge in den Auswahllisten zum Hinzufügen von Principals zu einer Zugriffssteuerungsliste (vgl. Dialog zur Änderung der Zugriffssteuerungsliste) maximal angezeigt werden Standardwert: 300
MAX_SHOWN_REFERENCES	Gibt an, wie viele Einträge in den Listen im Referenzen-Dialog des Content-Clients (Classic) maximal angezeigt werden Standardwert: 300
MAX_SHOWN_TEMPLATES	Gibt an, wie viele Einträge in der Auswahlliste <i>Vorlage</i> des Content-Clients (Classic) maximal angezeigt werden (vgl. Metadaten-Dialog und Dialog zum Anlegen eines neuen Objekts) Standardwert: 300
MAX_SHOWN_VERSIONS	Gibt an, wie viele Versionen im Dialog <i>Alte Version wiederherstellen</i> des Content-Clients (Classic) maximal angezeigt werden. Es werden die jeweils jüngsten Objektversionen angezeigt. Standardwert: 300

Application	
Parameter	Beschreibung
OPTION_SHOW_DEBUG_INFO	Wenn der Wert dieses Parameters true ist, werden in einigen Dialogen des Content-Clients (Classic) zusätzliche Informationen angezeigt. Diese Informationen können zur Fehlersuche benutzt werden. Standardwert: false
VipObjectApplication	
keine	keine
VipUserApplication	
extend-profile	Eine durch Leerzeichen separierte Liste von Repositories. Diese Repositories werden für die Erweiterung des Profils verwendet.
WebServiceApplication	
allowedWebsites	Eine durch Kommas separierte Liste von Website-Namen. Auf die angegebenen Websites kann über WCM WebServices zugegriffen werden. Durch Eingabe eines Sternchens (*) wird der Zugriff auf alle WCM-verwalteten Websites erlaubt. Standardwert: leer (kein Zugriff über WCM WebServices möglich)

Application	
Parameter	Beschreibung
deniedWebsites	Eine durch Kommas separierte Liste von Website-Namen. Auf die angegebenen Websites kann über WCM WebServices nicht zugegriffen werden. Durch Eingabe eines Sternchens (*) wird der Zugriff auf alle WCM-verwalteten Websites verboten. Ein Verbot hat Vorrang vor einer Erlaubnis. Standardwert: leer

Parameter einer Application löschen

Um einen Parameter aus der Konfiguration einer Application zu entfernen, markieren Sie gewünschten Parameter im rechten Fensterbereich. Wählen Sie dann *Parameter löschen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Löschen eines Parameters

Allgemeine Einstellungen von Applications

So bearbeiten Sie die Einstellungen einer Application:

1. Wählen Sie *Konfiguration* → *Applications*.
2. Markieren Sie die gewünschte Application im linken Baum.

Im rechten Fensterbereich werden die Einstellungen auf zwei Registern angezeigt.

- Die Angaben auf dem Register *Basisdaten* werden beim Anlegen der Application gemacht. Sie können nachträglich alle Einstellungen mit Ausnahme des Namens ändern.
 - Auf dem Register *Ressourcen* können Sie die Einstellungen für den Zugriff auf Java-Ressourcen ändern.
3. Nehmen Sie die gewünschten Änderungen vor.
 4. Klicken Sie auf die Schaltfläche *Übernehmen*.

Repository zu einer Application zuordnen

Über *Konfiguration* → *Applications* → {Application-Name} → *Repositories* erhalten Sie eine Übersicht über die Repositories, die dieser Application bereits zugeordnet sind. Die zugeordneten Repositories werden im rechten Fensterbereich in einer Liste angezeigt.

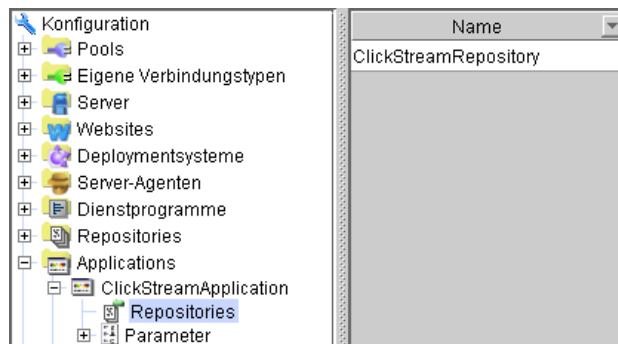


Abb. 19 – Zu einer Application zugeordnete Repositories

Zuordnung eines Repository zu einer Application herstellen

So ordnen Sie ein Repository zu einer Application zu:

1. Wählen Sie *Konfiguration* → *Applications* → *{Application-Name}* → *Repositories*.
2. Wählen Sie *Repository zuordnen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Zuordnen eines Repository zu einer Application

Der Dialog *Repository auswählen* wird geöffnet. Er zeigt eine Liste der noch nicht zugeordneten Repositories.

3. Markieren Sie das gewünschte Repository.
4. Klicken Sie auf die Schaltfläche *OK*.

Zuordnung eines Repository zu einer Application aufheben

Um die Zuordnung eines Repository zu einer Application wieder aufzuheben, markieren Sie das entsprechende Repository im rechten Fensterbereich. Wählen Sie *Zuordnung des Repository aufheben* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Aufheben der Zuordnung eines Repository zu einer Application

Struktur unterhalb des Elements *Applications*

Jede angelegte Application hat die Knoten *Repositories* und *Parameter*. Wenn Sie der Application Repositories oder Parameter zugeordnet haben, erscheinen diese Einträge unterhalb des jeweiligen Knotens (in der folgenden Abbildung wurde der Application *VipHCLApplication* ein Parameter zugeordnet).

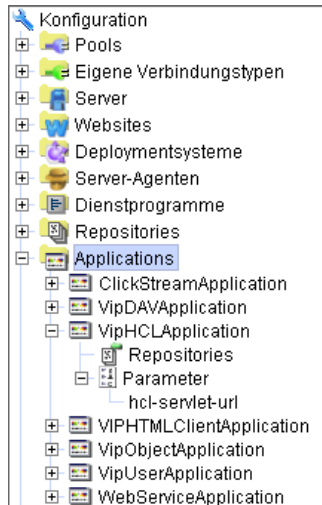


Abb. 20 – Struktur unterhalb des Elements *Applications*

Application löschen

So löschen Sie eine Application:

1. Wählen Sie *Konfiguration* → *Applications*.
2. Markieren Sie die gewünschte Application im linken Baum.
3. Wählen Sie *Application löschen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Löschen einer Application

4. Bestätigen Sie die Sicherheitsabfrage mit *Ja*.

Die Konfiguration der Application wird aus dem WCM-System entfernt, die dazugehörige Java-Klasse wird jedoch nicht gelöscht.

Application zu einem Content-Server zuordnen

Ein Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft, kann nur auf Applications zugreifen, die ihm zugeordnet sind. Über *Konfiguration* → *Server* → *{Name des Content-Servers}* → *Applications* erhalten Sie eine Übersicht über die Applications, die diesem Server bereits zugeordnet sind. Die Applications werden im rechten Fensterbereich in einer Liste angezeigt.

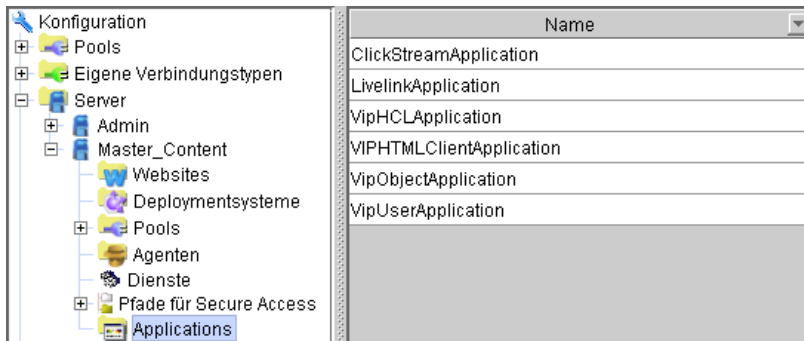


Abb. 21 – Zu einem Content-Server zugeordnete Applications

Zuordnung einer Application zu einem Content-Server herstellen

So ordnen Sie eine Application zu einem Content-Server zu, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft:

1. Wählen Sie *Konfiguration* → *Server* → *{Name des Content-Servers}* → *Applications*.
2. Wählen Sie *Application zuordnen* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Zuordnen einer Application zu einem Content-Server

Der Dialog *Application auswählen* wird geöffnet. Er zeigt eine Liste der noch nicht zugeordneten Applications.

3. Markieren Sie die gewünschte Application.
4. Klicken Sie auf die Schaltfläche *OK*.

Hinweis: Sie können demselben Content-Server nicht zwei Applications mit dem gleichen logischen Namen zuordnen.

Zuordnung einer Application zu einem Content-Server aufheben

Um die Zuordnung einer Application zu einem Content-Server wieder aufzuheben, markieren Sie die Application im rechten Fensterbereich. Wählen Sie *Zuordnung der Application aufheben* aus dem Kontextmenü, oder klicken Sie auf das entsprechende Symbol.



Symbol zum Aufheben der Zuordnung einer Application zu einem Content-Server

3.3 LDAP

Durch den Einsatz eines LDAP-Verzeichnisdienstes haben Sie die Möglichkeit, den Verwaltungsaufwand im Zusammenhang mit der Konfiguration und Pflege von Benutzerdaten erheblich zu reduzieren. Das Portal Manager API steht auf allen Content-Servern zur Verfügung, die im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server laufen. Diese Content-Server wiederum greifen auf die gemeinsame Datenhaltung des WCM-Systems zurück. Deshalb können die Benutzer-, Gruppen- und Rolleninformationen eines LDAP-Verzeichnisdienstes vom Portal Manager API übernommen und genutzt werden.

Detaillierte Informationen zur Konfiguration einer LDAP-Anbindung finden Sie im Livelink WCM Server-Installationshandbuch. Das Portal Manager API bietet außerdem die Möglichkeit, direkt auf den LDAP-Verzeichnisdienst zuzugreifen. Die LDAP-Unterstützung im Portal Manager API ist durch die Klasse `DirectoryRepository` gegeben.

Die Klasse `DirectoryRepository`

Die Klasse `DirectoryRepository` realisiert den Zugriff auf einen LDAP-Verzeichnisdienst. Hierbei handelt es sich um eine alternative, von der LDAP-Anbindung eines Administrationsservers unabhängige Art, auf LDAP zuzugreifen. Es ist möglich, über das `DirectoryRepository` Daten vom LDAP-Server zu laden und dort Daten zu speichern. Ein Anwendungsbeispiel hierzu finden Sie im Abschnitt "Benutzer über ein Portal in einem LDAP-System anlegen" auf Seite 140.

Das `DirectoryRepository` kann über Parameter konfiguriert werden. Die Konfiguration umfasst beispielsweise die Angaben eines Suchknotens (Base), ab dem neue Einträge eingefügt und vorhandene Einträge gesucht und geladen werden. Jeder Zugriff auf den LDAP-Verzeichnisdienst erfordert auch eine Authentifizierung des jeweiligen Benutzers. Um

neue Einträge einfügen zu können, muss der Benutzer Administratorrechte für den LDAP-Verzeichnisdienst haben.

Die Klasse `DirectoryRepository` kann für die Authentifizierung von Benutzern für das Portal Manager API benutzt werden, da sie das Interface `LoginRepository` implementiert. Wenn auf das WCM-System schreibend zugegriffen werden soll oder die angeforderten Objekte nicht für alle Benutzer lesbar sind, muss sich der Benutzer über das `VipUserRepository` anmelden. Bei dieser Anmeldung wird dem Benutzer eine Context-ID zugeordnet, die das WCM-System für die Rechteüberprüfung benötigt.

Damit ein `DirectoryRepository` für die Benutzerverwaltung im Portal Manager API benutzt werden kann, ist es notwendig, in den LDAP-Verzeichnisdienst zusätzliche Objektklassen und Attribute einzubinden. Es handelt sich um dieselben Einstellungen, die vorzunehmen sind, wenn Livelink WCM Server auf den LDAP-Verzeichnisdienst zugreift.

Es ist möglich, weitere Objektklassen und Attribute in den LDAP-Verzeichnisdienst einzubinden. Beim Hinzufügen eines Eintrags in LDAP können diese Objektklassen angegeben werden. Sie müssen als Value im `RepositoryEntry` unter dem Schlüssel `AttributeName.OBJECTCLASS` als `RepositoryMap` eingetragen werden.

Die Bezeichner für die Attributtypen im LDAP-Verzeichnisdienst weichen von den intern im Administrationsserver benutzten ab. Die Klasse `AttributeName` enthält Konstanten für die gültigen Attributnamen und Methoden, die bei der Umsetzung dieser Namen zwischen den internen Attributbezeichnern und der LDAP-Schreibweise helfen (Mapping).

Das Mapping ist nur für individuelle Implementationen einer auf LDAP-Verzeichnisdienste zugreifenden Repository-Klasse nötig. Wenn die Klasse `DirectoryRepository` Daten schreibt, werden die internen Attributbezeichner automatisch in die Bezeichner für den LDAP-Verzeichnisdienst umgewandelt. Beim Lesen aus dem LDAP-Verzeichnisdienst findet das dann auch automatisch in der umgekehrten Richtung statt.

Hinweis: Microsoft Active Directory unterstützt nicht das Überschreiben eines Eintrags. Deshalb führt bei Microsoft Active Directory die Methode `putEntry` zu einer Exception, wenn der Eintrag im LDAP-Verzeichnisdienst bereits vorhanden ist. Das Neuanlegen eines Eintrags ist jedoch möglich. Alle Methodenaufrufe finden im Kontext des aktuell an der Session angemeldeten Benutzers statt. Lediglich die Profilgewinnung beim Anmelden und das Neuanlegen eines Eintrags finden im Kontext von `principal` statt.

Wie jedes Repository muss auch das `DirectoryRepository` über den Admin-Client konfiguriert werden.

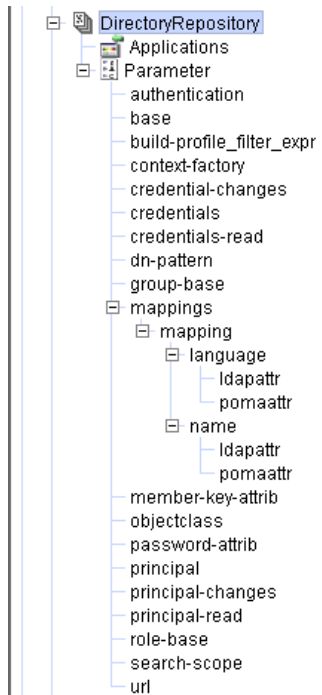


Abb. 22 – Das `DirectoryRepository` mit seinen Parametern

3.4 Fehlersuche

Die Fehlersuche innerhalb der Methoden und Klassen, die von der Klasse `Base` direkt oder indirekt ableiten, nennt man Tracing. Das Tracing-Verhalten können Sie in der Datei **`trace.properties`** konfigurieren. Damit die in der Datei vorgenommenen Änderungen wirksam werden, müssen die Server neu gestartet werden. Sie können das Tracing-Verhalten auch im Admin-Client konfigurieren; in diesem Fall ist kein Neustart der Server erforderlich.

Die Datei **`trace.properties`** enthält die folgenden Einträge:

Einträge in der Datei `trace.properties`

Eintrag	Bedeutung
<code>output.policy</code>	Mit diesem Eintrag wird die Art der Ausgabe festgelegt. Folgende Werte sind möglich: Stdout: Standardausgabekanal Sterr: Standardfehlerkanal Process: Eine Datei für alle Ausgaben (Standard) Thread: Eine Datei pro Thread Class: Eine Datei pro Klasse Thread+Class: Eine Datei pro Thread und Klasse
<code>output.overwrite</code>	Mit diesem Eintrag werden bestehende Protokolldateien überschrieben (Wert <code>true</code>) oder die Daten werden an bestehende Dateien angefügt (Wert <code>false</code>).
<code>output.limit</code>	Die maximale Größe der Ausgabedatei in KB
<code>file.path</code>	Der Pfad zu dem Verzeichnis, in dem sich die Protokolldateien befinden
<code>file.prefix</code>	Das Präfix der Dateinamen

Eintrag	Bedeutung
<code>file.suffix</code>	Das Suffix der Dateinamen (in der Regel die Dateinamenerweiterung)
<code>clearer.delay</code>	Die Zeit in Sekunden zwischen zwei Läufen, damit die Thread-Tabelle um nicht mehr laufende Threads bereinigt werden kann. Bei einem Wert von 0 wird kein Bereinigungs-Thread gestartet (nur für Tests, in einer Produktivinstallation muss immer ein Bereinigungs-Thread laufen). Die Zeit sollte mindestens 10 Sekunden betragen, empfohlen wird eine Zeitdauer von 60 bis 300 Sekunden.
<code>clearer.entries</code>	Die Mindestanzahl von Einträgen in der Thread-Tabelle, damit die Bereinigung dieser Tabelle überhaupt durchgeführt wird. Dieser Eintrag wird nur beachtet, wenn der Bereinigungs-Thread läuft. Es sollten 50 bis 200 Einträge in der Tabelle vorhanden sein.
<code>trace.filter</code>	Die erzeugten Thread-Objekte benutzen den hier angegebenen Filterwert, um zu entscheiden, welche Trace-Ausgaben zum Logging-Channel geschrieben werden. Der Filterwert wird als Bit-Feld interpretiert. Jedes Bit steht für eine Art von Trace-Information (z.B. Eintritt in eine Methode, Verlassen einer Methode, Warnung, Fehler). Der Wert 0 bewirkt, dass kein Tracing durchgeführt wird. Der Filterwert kann auch als Hex-Zahl angegeben werden, indem 0x vorangestellt wird (das Setzen aller Bits für bitorientierte Filterimplementationen im Logging-Channel kann mit dem Wert 0x7F erreicht werden). Detaillierte Informationen finden Sie im Kommentar der Datei trace.properties.
<code>trace.indentspaces</code>	Die Anzahl von Leerzeichen, die pro Einrückungstiefe hinter dem Datumsstempel ausgegeben werden soll.

Eintrag	Bedeutung
<code>trace.width</code>	Die Breite der Ausgabezeilen. Die Ausgabe wird an Wortgrenzen umgebrochen. Die Breite muss mindestens den Wert 40 betragen. Wenn der Text nicht umgebrochen werden soll, kann der Wert 0 angegeben werden.
<code>trace.timestamp</code>	Die Art des Zeitstempels am Anfang jeder Zeile. Einer der vier folgenden Werte ist erlaubt: NO: Kein Zeitstempel INTERNAL: Zeitstempel in Millisekunden SHORT: Zeitstempel Stunde bis Millisekunde LONG: Zeitstempel Jahr bis Millisekunde
<code>trace.format</code>	Der Format-String für die Trace-Ausgaben. Folgende Platzhalter sind möglich: {0}: Der vollständige Name der Klasse, die den Trace aufruft {1}: Der Name der Klasse ohne Package-Namen, die den Trace aufruft {2}: Der Name der Methode, aus der der Trace aufgerufen wird {3}: Der Name des Objekts, das den Trace benutzt {4}: Der Name des aktuellen Threads {5}: Die Einrückungstiefe als Zahl {6}: Der Trace-Text

Eintrag	Bedeutung
<code>trace.acceptclasses</code>	Grundsätzlich werden die Traces durch Überprüfen des Eintrags <code>trace.filter</code> gefiltert. Bei bestimmten Klassen bestehen Ausnahmen: Explizit benannte Klassen haben Filterwerte, die von den Standard-Filterwerten abweichen. Dieses Verhalten ist sehr zeitintensiv und kann daher (de)aktiviert werden. Durch den Wert <code>true</code> wird die Ausnahmebehandlung der entsprechenden Klassen aktiviert, durch den Wert <code>false</code> deaktiviert. Detaillierte Informationen finden Sie im Kommentar der Datei <code>trace.properties</code>.
<code>class1</code> bzw. <code>class2</code>	Mit diesem Eintrag wird angegeben, für welche Klasse(n) das Tracing eingeschaltet werden soll. Mögliche Werte sind <code>com.org.pkg.Class1</code>: Tracing für die Klasse <code>com.org.pkg.*</code>: Tracing für das Package <code>com.org.pkg.**</code>: Tracing für das Package mit allen Unterpackages

KAPITEL 4

Klassen und Interfaces des Portal Manager API

In diesem Kapitel werden Ihnen die Klassen vorgestellt, die für das Verständnis der Konzepte des Portal Manager API benötigt werden.

4.1 Klassendiagramme und Klassenbeschreibungen

Klassendiagramme

Den einzelnen Abschnitten dieses Kapitels sind zur Übersicht Klassendiagramme zugeordnet. Das Gesamtdiagramm finden Sie als Datei unter **{WCM-Installationsverzeichnis}\documentation\manual\portalManagerClassDiagram.pdf**. Die Diagramme im Text stellen die jeweils relevanten Ausschnitte daraus dar. Im Gesamtdiagramm sind die thematisch zusammengehörenden Klassen in einer bestimmten Farbe gezeichnet.

Für die Klassendiagramme wird das folgende – an UML-Diagramme angelehnte – Schema verwendet:

- *Weißer Kästen* bezeichnen Klassen oder Interfaces, die aus anderen Teilen stammen. Andere Teile können Klassen aus anderen Themenbereichen des Portal Manager API sein, aber auch solche, die aus Java-Basisbibliotheken stammen (Java Foundation Classes oder Java Extensions). Diese Klassen werden im Text nicht erläutert. Sie sind in das Diagramm aufgenommen, um kontextuelle Bezüge herzustellen.
- *Hellere Kästen* symbolisieren Interfaces. Der Name des Interface ist immer kursiv geschrieben.
- *Dunklere Kästen* stehen für Klassen. Nur die Namen abstrakter Klassen sind kursiv geschrieben.

Die Beziehungen zwischen den Klassen und Interfaces wird durch Pfeile dargestellt. Ein gefüllter Pfeil steht für die Vererbung einer Klasse (Schlüsselwort *extends*), ein hohler Pfeil für die Implementation eines Interfaces (Schlüsselwort *implements*). Gestrichelte Pfeile stellen Assoziationsbeziehungen zwischen Klassen dar. In diesem Fall wird die Beziehung explizit benannt.

Die folgende Abbildung verdeutlicht die verschiedenen Darstellungsformen:

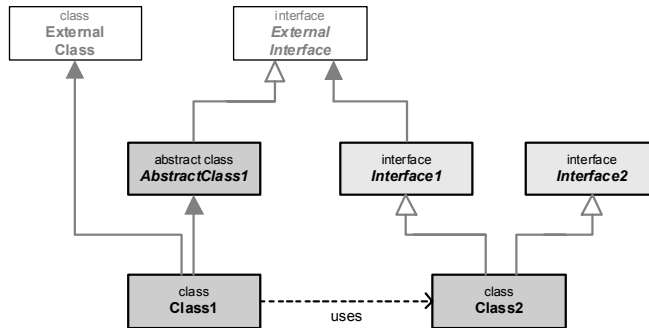


Abb. 23 – Darstellungsschema für die Klassendiagramme

Klassenbeschreibung

Mit der Installation des WCM-Systems wird die komplette Dokumentation der Klassen des Portal Manager API als Javadoc-Seiten installiert. Sie finden die Javadoc-Dokumentation im Verzeichnis **{WCM-Installationsverzeichnis}\documentation\javadoc**. Die Klassenbeschreibungen in diesem Handbuch unterscheiden sich von der Javadoc-Dokumentation. Die Javadoc zum Portal Manager API enthält ausführliche Informationen zu sämtlichen Klassen und Interfaces sowie die dazugehörigen Methoden.

4.2 Grundlegende Klassen

Alle mit dem Portal Manager API ausgelieferten Klassen verwenden eine Reihe von grundlegenden Klassen, die sich in Gruppen zusammenfassen lassen:

- Basisdatentypen
- Filterung und Sortierung
- RepositoryMap und zugehörige Klassen
- Repositories

Diese vier Gruppen werden in diesem Abschnitt beschrieben. Zur Übersicht ist jeweils ein Klassendiagramm abgebildet, das den entsprechenden Ausschnitt aus dem Gesamtdiagramm zeigt.

Die grundlegenden Klassen liefern den wichtigen algorithmischen und datentechnischen Unterbau des Portal Manager API. Darin sind so wesentliche Eigenschaften wie Vergleichbarkeit, Filterbarkeit und Sortierbarkeit inbegriffen. Die höchste Abstraktionsstufe wird durch die Repositories erreicht. Da alle grundlegenden Klassen direkt oder indirekt in Session-Beans eingesetzt werden, ist die Kenntnis dieser Klassen und der damit verfolgten Konzepte wichtig.

Basisdatentypen

Alle Attributschlüssel eines WCM-Objekts implementieren das Interface Key. Die Attributwerte implementieren das Interface Value. Dieses Interface erweitert – wie in Abbildung 25 “Die grundlegenden Klassen im Gesamtklassendiagramm” auf Seite 105 dargestellt – die Interfaces Comparable, Cloneable und Serializable.

Die Objektdaten können als Attribut-Wert-Paare dargestellt werden und werden so vom Portal Manager API verwaltet. Weitere Informationen zu diesem Thema finden Sie im WCM Java API-Programmierhandbuch.

Schlüsselwerte (Keys)

Da sowohl die Klasse `TupleMap` als auch die Klasse `RepositoryMap` im Wesentlichen die Zuordnungen von Schlüssel zu Datenwerten beschreiben, wurde ein Interface `Key` eingeführt. Es können aber beliebige Klassen als Schlüssel in den Repositories benutzt werden, sofern sie das Interface `Key` implementieren.

Datenwerte (Values)

Die Werte, die gespeichert werden, müssen dem Interface `Value` entsprechen. Die Basisdatenimplementationen, die mit dem Portal Manager API ausgeliefert werden, tragen bis auf einen Fall als Suffix im Klassennamen die Bezeichnung "Value". Auch hier können andere als die ausgelieferten Klassen benutzt werden, solange sie das Interface `Value` berücksichtigen.

Das Interface Value

Vollständiger Name: `de.gauss.lang.Value`

Das Interface `Value` repräsentiert einen Datenwert. Datenwerte werden in der Klasse `TupleMap` eingesetzt, allerdings nicht als Schlüssel (dazu wurde das Interface `Key` vorgesehen, das das Interface `Value` noch erweitert). Ein Datenwert muss vergleichbar, klonbar und serialisierbar sein, daher basiert das Interface `Value` auf den drei entsprechenden Interfaces.

Herauszustellen ist eine besondere Implementation des Interface Value: die Klasse TupleMap. Eine TupleMap speichert Paare von Key und Value; da sie selbst vom Typ Value ist, kann sie wiederum als Wert in einer anderen TupleMap auftreten. So entstehen mehrstufige (rekursive) Strukturen. Die folgende Abbildung veranschaulicht diese Konstruktion.

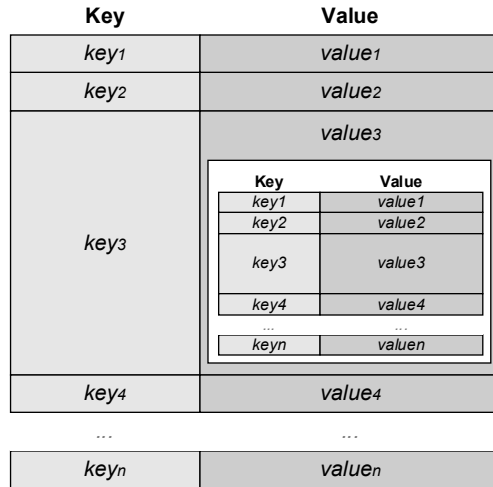


Abb. 24 – Mehrstufige Schlüssel-Wert-Listen in der TupleMap

Es ist natürlich auch möglich, dass eine TupleMap sich selbst enthält, auch wenn das eine nicht unbedingt sinnvolle und gewollte Konstellation ist. Einige Algorithmen in den Klassen des Portal Manager API, die auf TupleMap operieren, würden in diesem Fall in den Zustand einer Endlosrekursion gelangen.

Klassendiagramm

Die folgende Abbildung zeigt den relevanten Ausschnitt aus dem Gesamtklassendiagramm.

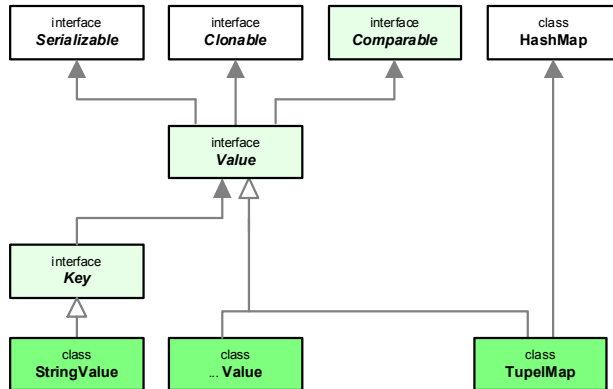


Abb. 25 – Die grundlegenden Klassen im Gesamtklassendiagramm

Filterung und Sortierung

Das Filtern und Sortieren von Daten sind grundsätzliche Funktionen, die häufig in der Verarbeitung von Datenbeständen benötigt werden.

Filterung

Eine Filterung wird allgemein durch das Interface `Filter` (vollständiger Name: `de.gauss.vip.lang.filter.Filter`) beschrieben, für das mehrere Implementationen bereitgestellt sind. Komplexe Filter können aus einfachen Filtern nach dem Baukastenprinzip zusammengefügt werden. Detaillierte Informationen zum Thema Filterung finden Sie im WCM Java API-Programmierhandbuch.

Sortierung

Eine Sortierung wird durch das Interface `Sort` (vollständiger Name: `de.gauss.lang.Sort`) beschrieben. Die einzige Implementierung des Interface `Sort` stellt die Klasse `KeySort` dar. Diese Klasse wird dazu verwendet, ein Sortierkriterium für die in einer `RepositoryMap` enthaltenen `RepositoryEntry`-Objekte zu definieren (siehe Abschnitt “`RepositoryMap` und zugehörige Klassen” ab Seite 107). Die Klasse `KeySort` enthält hierzu einen Schlüssel und eine Sortierreihenfolge. Der Schlüssel bestimmt das Sortierkriterium der `RepositoryEntry`-Objekte. Zum Vergleich der `RepositoryEntry`-Objekte wird der Wert herangezogen, den die `RepositoryEntry`-Objekte für den angegebenen Schlüssel enthalten.

Beispiel

Eine `RepositoryMap` enthält drei Objekte der Klasse `RepositoryEntry`. Der erste `RepositoryEntry` enthält für den Schlüssel “A” den Wert “Berta”, der zweite `RepositoryEntry` enthält für diesen Schlüssel den Wert “Cäsar”, der dritte den Wert “Anton”. Wenn man den Schlüssel “A” angibt, ordnet die Klasse `KeySort` die `RepositoryEntry`-Objekte in der folgenden Reihenfolge: dritter `RepositoryEntry`, erster `RepositoryEntry`, zweiter `RepositoryEntry`.

Ein Beispiel für die Verwendung der Klasse `KeySort` finden Sie im Code des Abschnitts “Dynamische Navigationselemente erzeugen” ab Seite 150.

Klassendiagramm

Die folgende Abbildung zeigt den relevanten Ausschnitt aus dem Gesamtklassendiagramm.

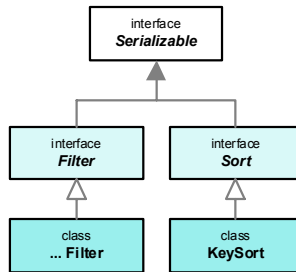


Abb. 26 – Die Filterungs- und Sortierungsklassen im Gesamtklassendiagramm

RepositoryMap und zugehörige Klassen

Die Klasse RepositoryMap zählt zu den universellsten der grundlegenden Datentypen im Portal Manager API. Die zu RepositoryMap gehörenden Klassen dürfen nicht mit den Klassen verwechselt werden, die das Interface Repository implementieren. Obwohl sie thematisch eng miteinander verknüpft sind, besteht zwischen ihnen keine Vererbungsrelation.

Klassendiagramm

Die folgende Abbildung zeigt den relevanten Ausschnitt aus dem Gesamtklassendiagramm.

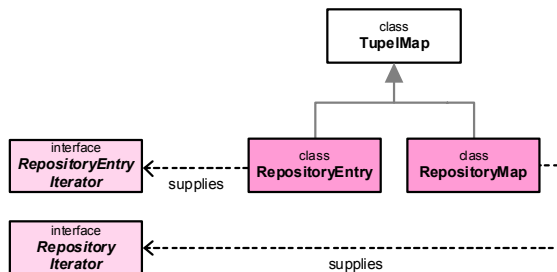


Abb. 27 – Die Klasse RepositoryMap im Gesamtklassendiagramm

In der folgenden Abbildung sind die Klassen zusammengetragen, mit denen im Portal Manager API die Datenspeicherung durchgeführt wird. Das Diagramm stellt nicht die Vererbungshierarchie in den Vordergrund, sondern die strukturellen Beziehungen.

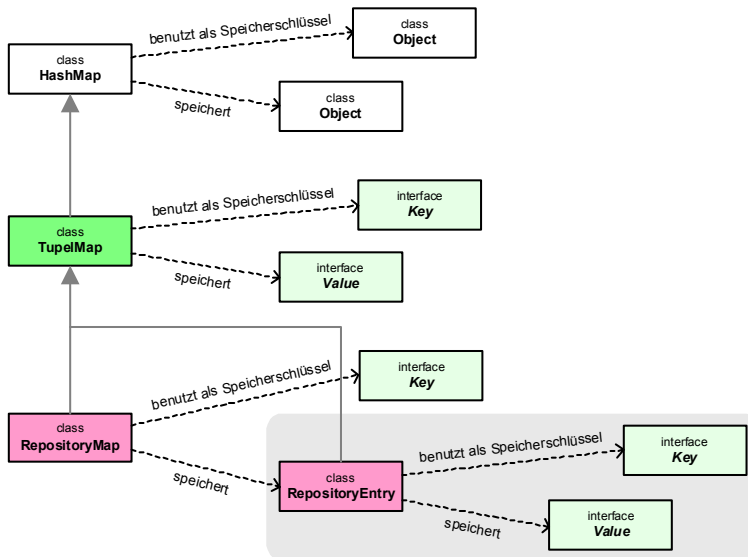


Abb. 28 – Strukturelle Beziehungen der komplexen Datentypen

Die Klasse RepositoryEntry

Vollständiger Name: `de.gauss.vip.repository.RepositoryEntry`

Objekte der Klasse `RepositoryEntry` sind die Werte, die in einer `RepositoryMap` gespeichert werden. Ein `RepositoryEntry` ist im Wesentlichen auch eine Map, da die Klasse von der Klasse `TupleMap` abgeleitet ist. Durch Speicherung eines Objekts dieser Klasse in einer `RepositoryMap` entsteht eine zweistufige Datenstruktur.

Eine Eigenschaft der Klasse `RepositoryEntry` gegenüber der Klasse `TupleMap` besteht darin, eine Schnittstelle zu einem Directory-Service

via JNDI zu liefern. Ein `RepositoryEntry` kann sowohl mit einem JNDI-Datenbestand initialisiert werden, wie auch sich selbst als JNDI-Datenbestand präsentieren. Die entsprechende Schnittstelle ist das JNDI-Attributes-Interface.

Das Interface `RepositoryEntryIterator`

Vollständiger Name:

`de.gauss.vip.repository.RepositoryEntryIterator`

Ein `RepositoryEntryIterator` ist eine Aufzählung über die Schlüssel-Datenwert-Paare in einem `RepositoryEntry`. Ein neuer Iterator steht per Definition vor dem ersten Paar. Um auf die Paare zugreifen zu können, muss der Iterator für das erste Paar und die darauf folgenden Paare mit einer der `next`-Methoden weitergerückt werden.

Die Klasse `RepositoryMap`

Vollständiger Name: `de.gauss.vip.repository.RepositoryMap`

Die Klasse `RepositoryMap` erweitert die Klasse `TupleMap` insbesondere um die Möglichkeit, Daten zu filtern, zu sortieren und aufzuzählen. Statt jedoch Werte vom Typ `Value` zu speichern, werden Werte vom Typ `RepositoryEntry` gespeichert. Da ein `RepositoryEntry` auch von `TupleMap` abgeleitet ist, ergeben sich zwangsläufig mehrstufige Listen. Dies ist in der folgenden Abbildung dargestellt.

Key	RepositoryEntry														
key ₁	entry ₁														
key ₂	entry ₂														
key ₃	entry ₃														
	<table><tr><th>Key</th><th>Value</th></tr><tr><td>key₁</td><td>value₁</td></tr><tr><td>key₂</td><td>value₂</td></tr><tr><td>key₃</td><td>value₃</td></tr><tr><td>key₄</td><td>value₄</td></tr><tr><td>...</td><td>...</td></tr><tr><td>key_n</td><td>value_n</td></tr></table>	Key	Value	key ₁	value ₁	key ₂	value ₂	key ₃	value ₃	key ₄	value ₄	...	...	key _n	value _n
	Key	Value													
	key ₁	value ₁													
	key ₂	value ₂													
	key ₃	value ₃													
	key ₄	value ₄													
...	...														
key _n	value _n														
key ₄	entry ₄														
...	...														
key _n	entry _n														

Abb. 29 – Mehrstufige Schlüssel-Wert-Listen in der RepositoryMap

Zum Filtern, Sortieren und Aufzählen der Daten einer RepositoryMap existiert ein Iterator (RepositoryIterator), mit dem ein zuvor gefilterter oder sortierter Datenbestand durchlaufen werden kann.

Das Interface RepositoryIterator

Vollständiger Name:

de.gauss.vip.repository.RepositoryIterator

Ein RepositoryIterator ist eine Aufzählung über die Schlüssel-Datenwert-Paare in einer RepositoryMap, in denen die Datenwerte immer vom Typ RepositoryEntry sind. Ein neuer Iterator steht per Definition vor dem ersten Paar. Um auf die Paare zugreifen zu können, muss der Iterator für das erste Paar und die darauf folgenden Paare mit einer der next-Methoden weitergerückt werden.

Repositories

Zugriff auf externe Ressourcen

Jeder Zugriff auf externe Ressourcen (*Anwendungsdatenbestände*) erfordert einerseits die Definition der Ressource (wie etwa Hostname, Portnummer, Anwendungsnamen) und andererseits Autorisierungsinformationen für den Zugriff (Benutzername und Passwort).

Im Portal Manager API sind Anwendungsdatenbestände durch Repositories modelliert. Die Initialisierungsdaten zur Definition der Ressource (URL, Benutzerkennung, Passwort und diverse Einstellungsparameter) werden einem Repository als Parameter übergeben. Die erwarteten Einträge sind für jede Anwendung unterschiedlich.

Optionen bei der Autorisierung

Sofern eine Anwendung eine Autorisierung erfordert, können hierzu zwei verschiedene Wege gewählt werden: Autorisierungsdaten können entweder *direkt* übergeben oder *zentral* verwaltet werden.

Wie bereits beschrieben, können Autorisierungsdaten direkt in den Parametern übergeben werden. Die andere Möglichkeit ist, diese Verwaltung zentral mithilfe von Account-Informationen (Login-Informationen) durchzuführen, die im Portal Manager API gespeichert sind. Die Account-Informationen sind zentral und können – falls es mehrere Repositories gibt – von allen Repositories benutzt werden. Hierdurch entsteht ein Single-Sign-On-Mechanismus.

Systeme, die Account-Informationen speichern, können ebenfalls als Applications interpretiert werden. Deshalb liegt es nahe, diese auch in Repositories zugänglich zu machen. Genau das ist im Portal Manager API umgesetzt.

Repository-Typen

Repositories werden für zwei Zwecke verwendet:

- Repräsentation einer Anwendung bzw. des Datenbestands einer Anwendung (primärer Einsatz im Portal Manager API)
- Ermittlung und Prüfung von benutzerspezifischen Account-Informationen (Login-Repositories)

Für den Sonderfall, dass gerade die Account-Informationen selbst den Datenbestand einer Anwendung darstellen, kann eine Repository-Implementation natürlich für beide Zwecke eingesetzt werden.

Login-Repositories

Login-Repositories sind Repositories, mit denen Account-Informationen verarbeitet werden. Im Portal Manager API gibt es zwei Login-Repositories, die die folgenden Datenquellen unterstützen:

- Account-Informationen in einem LDAP-Server (DirectoryRepository)
- Account-Informationen im Admin-Client (VipUserRepository); der Administrationsserver kann dazu auch auf den LDAP-Server zugreifen.

Jedes Login-Repository benutzt als Schlüssel die Benutzernamen. Als Datenwerte dienen alle Account-Informationen zu den jeweiligen Benutzern. Diese sind in einem RepositoryEntry abgelegt. In dieser Eigenschaft gleichen Login-Repositories den "normalen" Repositories.

Nach erfolgreicher Authentifizierung eines Benutzers wird sein Passwort gespeichert. Damit können andere Repositories das Passwort noch einmal verwenden, ohne dass es vom Benutzer erneut eingegeben werden muss (Single-Sign-On-Unterstützung).

Klassenhierarchie

Ein Repository wird durch das Interface `Repository` dargestellt. Das Interface `Repository` beinhaltet im Wesentlichen Zugriffsmethoden, die eine Implementierung mittels einer `RepositoryMap` nahelegen (jedoch nicht erzwingen). Einige der Implementationen des Interface `Repository` speichern die Repository-Daten tatsächlich als `RepositoryMap`, andere leiten die Methodenaufrufe an die dahinter stehende externe Datenquelle weiter.

Das Interface `Repository` wird von der abstrakten Basisklasse `RepositoryImpl` implementiert. Diese Klasse ist Grundlage für alle mit dem Portal Manager API ausgelieferten Repository-Implementationen.

Die Klasse `RepositoryImpl`

Vollständiger Name:

`de.gauss.vip.portalmanager.repository.RepositoryImpl`

Diese `Repository`-Implementation ist die abstrakte Basisimplementa-tion eines Repository für alle mit Livelink WCM Server ausgelieferten Repository-Klassen.

Ein Login-Repository ist als Interface `LoginRepository` modelliert. Es erweitert das Interface `Repository` um Methoden zur Prüfung der Anmeldung.

Sessions

Bei jedem Zugriff auf Repository-Inhalte muss eine Session angegeben werden. Die in der Session enthaltenen Informationen können vom Repository genutzt werden.

Eine Session beschreibt den aktuellen Verarbeitungskontext zwischen Browser und Webserver. Darüber hinaus enthält die Session eine ID. Über diese ID ist der Name des Benutzers verfügbar, der sich für die Session angemeldet hat. Dieser Benutzername sollte verwendet werden, um Account-Informationen bei einem Login-Repository abzufragen.

Das Interface `Session` ist im Zusammenhang mit den sessionbezogenen Klassen dargestellt (siehe Abbildung 32 “Die Bean-Klassen im Gesamtklassendiagramm” auf Seite 121).

Klassendiagramm

Die folgende Abbildung zeigt den relevanten Ausschnitt aus dem Gesamtklassendiagramm.

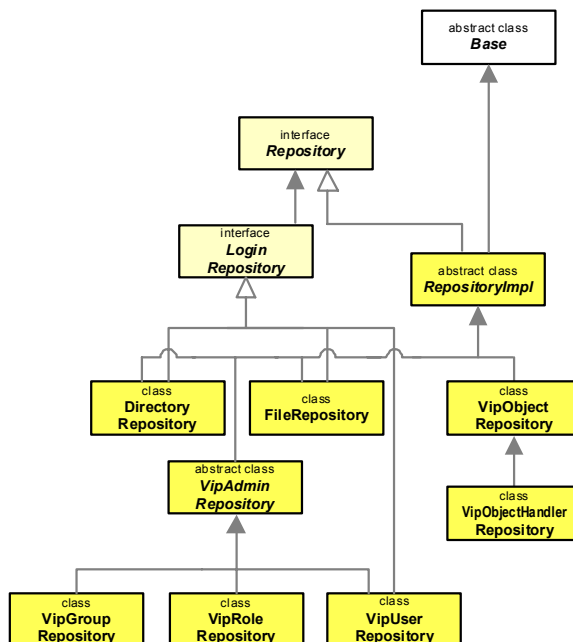


Abb. 30 – Die Repository-Klassen im Gesamtklassendiagramm

4.3 Application- und Bean-Klassen

Basisklasse

Bean-, Application- und Repository-Klassen sind von der Basisklasse `Base` abgeleitet. Diese Klasse implementiert mehrere Interfaces und erweitert selbst eine externe Klasse `CoreObject`.

Die Klasse `Base`

Vollständiger Name: `de.gauss.vip.portalmanager.core.Base`

`Base` ist eine abstrakte Basisklasse aller Applications und Repositories. Außerdem sind in der Klasse `Base` die Methoden zum Laden von Properties- und Multiproperties-Dateien enthalten.

`Base` enthält zusätzlich die statische Methode `startPMServer`, mit der Sie Content-Server, die im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server laufen, starten können. Wenn diese Content-Server bereits gestartet wurden, bleibt ein Aufrufen der Methode ohne Auswirkung. Die folgenden Parameter müssen als System-Properties gesetzt sein.

Tabelle 5 – Startparameter für Content-Server, die im Kontext einer JSP-Engine laufen

Parameter	Bedeutung
<code>vip.server.name</code>	Der Name des Content-Servers, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft
<code>vip.admin.host</code>	Der Name des Rechners, auf dem der Master-Admin-Server läuft
<code>vip.admin.socket</code>	Der Socketport des Master-Admin-Servers
<code>vip.admin.http</code>	Der HTTP-Port des Master-Admin-Servers

Parameter	Bedeutung
<code>vip.admin.secure</code>	Gibt an, ob die Kommunikation mit dem Master-Admin-Server verschlüsselt erfolgen soll. Mögliche Werte sind: <code>true</code> – verschlüsselte Kommunikation <code>false</code> – unverschlüsselte Kommunikation
<code>vip.installdir</code>	Das Installationsverzeichnis des WCM-Systems

Applications

Anwendungen werden durch das Interface `Application` beschrieben. Das Interface `Application` wird von der abstrakten Basisklasse `ApplicationImpl` implementiert. Auf dieser Basis werden für die Integration von Livelink WCM Server zwei konkrete Implementationen des Interface `Application` bereitgestellt. Diese beinhalten jedoch keine weiteren Methoden. Neue, das Interface `Application` implementierende Klassen für andere externe Anwendungen sollten ebenfalls von der abstrakten Basisklasse abgeleitet werden.

Klassendiagramm

Die folgende Abbildung zeigt den relevanten Ausschnitt aus dem Gesamtklassendiagramm.

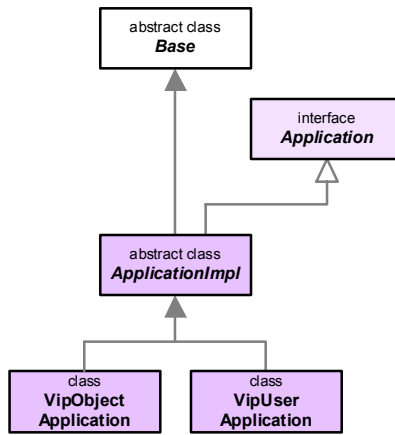


Abb. 31 – Die Application-Klassen im Gesamtklassendiagramm

Das Interface Application

Vollständiger Name:

`de.gauss.vip.portalmanager.application.Application`

Das Interface Application repräsentiert im Portal Manager API eine Anwendung. Durch dieses Interface ist insbesondere vorgegeben, dass eine Anwendung auf einem oder mehreren Repositories beruht, die den Zustand und den Datenbestand einer Anwendung beschreiben.

Zumindestens die folgenden Eigenschaften müssen von jeder Application-Implementation unterstützt werden:

- Eine Application soll möglichst das Factory-Design-Pattern erfüllen: Jede Application-Klasse kontrolliert, welche Instanzen existieren.

- Eine `Application` kann beliebig viele Ressourcen haben, die zur Konfiguration (z.B. zur Spracheinstellung) dienen. Der jeweilige Zweck einer Ressource ist von der konkreten `Application` abhängig. Ressourcen werden als Strings angegeben, die über den Klassenpfad identifizierte Resource-Bundle-Dateien im Dateisystem oder in JAR-Files bezeichnen. Ressourcen werden über den Standard-Class-Loader geladen.
- Eine `Application` benutzt eine beliebige Anzahl von Repositories, die sich durch den Namen unterscheiden.
- Eine `Application` hat ein Standard-Locale, das für Lokalisierungs- und Internationalisierungszwecke benutzt werden kann.

Die genannten Eigenschaften jeder `Application`-Implementation werden als Methoden in der Interface-Deklaration umgesetzt. Es gibt jedoch nur Methoden, um die Eigenschaftswerte zu ermitteln (`get`-Methoden), die Definition der Eigenschaften (durch `set`-Methoden) wird von einer `Application` nicht verlangt. Üblicherweise werden die Eigenschaften implizit über Konventionen oder durch Properties gesetzt, die bei der Initialisierung durch die `open`-Methode anzugeben sind.

Die Klasse `ApplicationImpl`

Vollständiger Name:

`de.gauss.vip.portalmanager.application.ApplicationImpl`

Eine `ApplicationImpl` ist eine abstrakte Implementation des Interface `Application`, auf dessen Basis ohne großen Aufwand konkrete `Application`-Klassen erstellt werden können. `ApplicationImpl` hat keinen öffentlichen Konstruktor. Über die in `Application` vorgegebenen Methoden hinaus existiert nur noch die Methode `getInstance` als Ersatz für einen Konstruktor, womit das Factory-Pattern umgesetzt wird.

Durch das erste Aufrufen der Methode `getInstance(String)` startet `ApplicationImpl` automatisch den Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft. Wenn Sie den Content-Server explizit starten möchten, benutzen Sie die Methode `startPMServer` der Klasse `Base`.

Die Klasse `ApplicationImpl` benutzt zur eigenen Initialisierung folgende Konventionen:

- Die Einstellungen der Application werden aus der Konfigurationsdatei **`application.xml`** gelesen. In der Konfigurationsdatei **`server.xml`** ist ein Mapping der logischen auf die existierenden Application-Namen enthalten.
- Jedes Repository, das `ApplicationImpl` benötigt, wird durch die Einstellungen aus der Konfigurationsdatei **`repository.xml`** gelesen.

In diesen Konfigurationsdateien sind alle Informationen enthalten, die es ermöglichen, `Application`-Objekte als Factory zu erzeugen. Dadurch wird die Versorgung mit Initialisierungsdaten bei konkreten `Application`-Implementationen erheblich vereinfacht.

Beans

Die Beans, die im Portal Manager API zur Verfügung gestellt werden, können als Instanzen in JSP-Skripten eingesetzt werden. Mit ihnen können beliebige Anwendungen über HTML-Seiten bedient werden. Als Scope für die Beans sollte "Session-Scope" eingestellt sein. Beans bedienen sich einer Application, die die eigentliche Anwendung repräsentiert. Die oberste aller Bean-Klassen heißt aus diesem Grund `SessionApplication`.

Hinweis: Alle Beans des Portal Manager API wurden für den Einsatz mit Session-Scope entwickelt. Bei Verwendung anderer Scopes – insbesondere “Request” und “Page” – besteht die Gefahr, viele CA-Lizenzen (CA = concurrent author) zu verbrauchen, die erst wieder freigegeben werden, wenn die zugehörigen Context-IDs in Livelink WCM Server ungültig werden.

Beans enthalten High-Level-Methoden auf Basis der `Application`- und `Repository`-Klassen. Diese Methoden fassen typische Einsatzfälle einer Anwendung auf einfach benutzbare und effiziente Weise zusammen.

Die abstrakte Implementation in `SessionApplication` sieht insbesondere bereits einfache Möglichkeiten zur Internationalisierung von Anwendungen vor. So können Benutzer unterschiedlicher Herkunft eine Anwendung in landesüblicher Sprache und mit landesüblichen Normen bedienen. Voraussetzung hierfür ist, dass Lokalisierungsinformationen über den entsprechenden Benutzer vorliegen.

Klassendiagramm

Die folgende Abbildung zeigt den relevanten Ausschnitt aus dem Gesamtklassendiagramm.

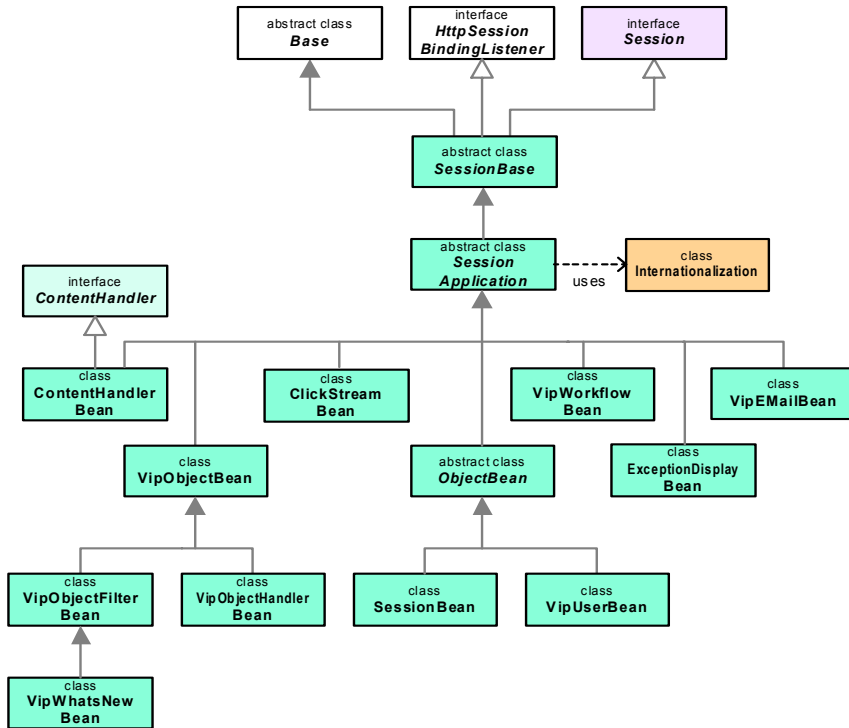


Abb. 32 – Die Bean-Klassen im Gesamtklassendiagramm

Die Klasse SessionApplication

Vollständiger Name:

de.gauss.vip.portalmanager.application.SessionApplication

Die Klasse SessionApplication stellt die abstrakte Basisklasse für alle Beans im Portal Manager API dar. Die Klasse stellt den Beans die folgende Funktionalität zur Verfügung:

- Zuordnung des Bean zu einer Session
- Methode zum Erhalten der zugeordneten Application
- Lokalisierungsinformationen
- Methoden zum Erhalten und Setzen des Repository-Namens

Das Standard-Locale und die Ressourcen für die jeweilige Application werden aus der Konfigurationsdatei **application.xml** ermittelt.

Die Klasse `Internationalization`

Vollständiger Name: `de.gauss.vip.i18n.Internationalization`

Die Klasse `Internationalization` übernimmt die Internationalisierung von Messages. Die Internationalisierung wird durch ein `Locale`-Objekt gesteuert.

Die Basis für die Internationalisierung sind Resource-Bundles. Ein `Internationalization`-Objekt kann beliebig viele Resource-Bundles auf Basis beliebiger `Locale`-Objekte benutzen. Der Zugriff auf Ressourcen wird wie üblich über Schlüssel durchgeführt, wobei die Auswahl der `Locale`-Objekte über das aktuelle `Locale` der Session gesteuert wird. Ein Schlüssel wird nacheinander für jedes für das `Locale` in Frage kommende Resource-Bundle geprüft, bis eines der Resource-Bundles einen Wert liefert. Wenn Resource-Bundles gleiche Schlüssel haben, wird das Resource-Bundle genommen, das in der Datei **application.xml** zuerst eingetragen ist.

Die Klasse `Internationalization` stellt Methoden zur Verfügung, um

- lokalisierte Messages für den Schlüssel zu erhalten
- `Locale`-spezifische Datums- und Zahlenformate zu erhalten

Die Klasse `SessionBean`

Vollständiger Name: `de.gauss.vip.portalmanager.SessionBean`

Ein `SessionBean` verwaltet Benutzerprofile auf Grundlage der eindeutigen Session-ID. (Die Bezeichnung Session-Bean bezieht sich auf ein Bean, das eine sessionbezogene Benutzerverwaltung durchführt. Dies sollte jedoch nicht mit den Session-Beans verwechselt werden, die Teil von Enterprise JavaBeans sind.) Eine Session-ID kann entweder als String oder als HTTP-Request angegeben werden.

Bevor ein Benutzer durch einen Anmeldevorgang authentifiziert wird, wird er nur als anonym Benutzer mit stark eingeschränkten Rechten geführt. Benutzer werden durch die Klasse `SessionBean` nicht automatisch angemeldet. Es ist möglich, einen Benutzer explizit abzumelden.

Das Interface `VipProfile`

Vollständiger Name: `de.gauss.vip.portalmanager.VipProfile`

Mithilfe des Interface `VipProfile` lassen sich zusätzliche Daten zu einem Benutzer verwalten und speichern. Der Content-Client verwendet `VipProfile` zum permanenten Speichern der Konfigurationsdaten eines Benutzers. Außerdem können WCM-unabhängige Daten wie z.B. die Adresse des Benutzers über `VipProfile` verwaltet und gespeichert werden.

Das `VipProfile` zu einem Benutzer kann durch das `SessionBean` über den Aufruf der Methode

`SessionBean.getCurrentUserProfile(String)` ermittelt werden; dabei ist die Benutzerkennung zu übergeben. Die Daten werden innerhalb von `VipProfile` als Schlüssel-Wert-Paare verwaltet und gegebenenfalls gespeichert, d.h., es kann unter einem bestimmten Schlüssel ein Wert (als Value) abgelegt werden.

Die über `VipProfile` zugreifbaren Daten sind in drei Abschnitte unterteilt:

- **transiente Profildaten** (`VipProfile.TRANSIENT`)

Dieser Abschnitt dient zur Speicherung von Profildaten, die während der Session eines Benutzers verfügbar sein müssen, aber nicht persistent abgelegt werden sollen. Die transienten Profildaten aus `VipProfile` können gelesen und manipuliert werden, ohne dass der jeweilige Benutzer im WCM-System angemeldet sein muss.

- **Client-bezogene Profildaten** (`VipProfile.KNOWN_USER`)

Dieser Abschnitt dient zu Verwaltung und Speicherung von Daten, die persistent zu einem Benutzer gespeichert werden sollen. Die Speicherung der Daten erfolgt dabei in einem Cookie. Demzufolge stehen die in diesem Abschnitt gespeicherten Daten nur auf dem Client (Browser) zur Verfügung, auf dem der Cookie abgelegt ist. Die Lebensdauer eines Cookie ist standardmäßig auf 90 Tage begrenzt, kann aber über den Parameter `KNOWNUSER_COOKIE_DURATION` der Klasse `VipUserApplication` geändert werden.

- **benutzerspezifische Profildaten** (`VipProfile.VIP_USER`)

Dieser Abschnitt dient zur persistenten Verwaltung und Speicherung von Benutzerdaten, die systemweit eindeutig verfügbar sein müssen. Die Speicherung der Profildaten dieses Abschnitts erfolgt abhängig von der Art der Benutzerverwaltung in einer Datenbank, einem LDAP-Verzeichnisdienst oder in Livelink. Um in diesem Abschnitt Daten auslesen oder manipulieren zu können, muss der jeweilige Benutzer am WCM-System angemeldet sein.

Innerhalb dieses Abschnitts müssen Daten in unterschiedlichen, mit eindeutigen Namen versehenen Unterprofilen gespeichert und verwaltet werden. Die Methoden `VipProfile.getProfileValue(String profileName, String attribute)` und `VipProfile.setProfileValue(String`

profileName,String attribute,Value value) erlauben den Zugriff und die Manipulation von Daten innerhalb dieses Abschnitts unter Angabe eines Profilnamens. Alternativ dazu kann über die Methode `VipProfile.setCurrentProfileName(String name)` der Name des für den Abschnitt `VIP_USER` zu benutzenden Profils gesetzt werden. Über die Methode `VipProfile.getCurrentProfileName` kann der aktuelle Name des für die benutzerspezifischen Profildaten benutzten Profils erfragt werden

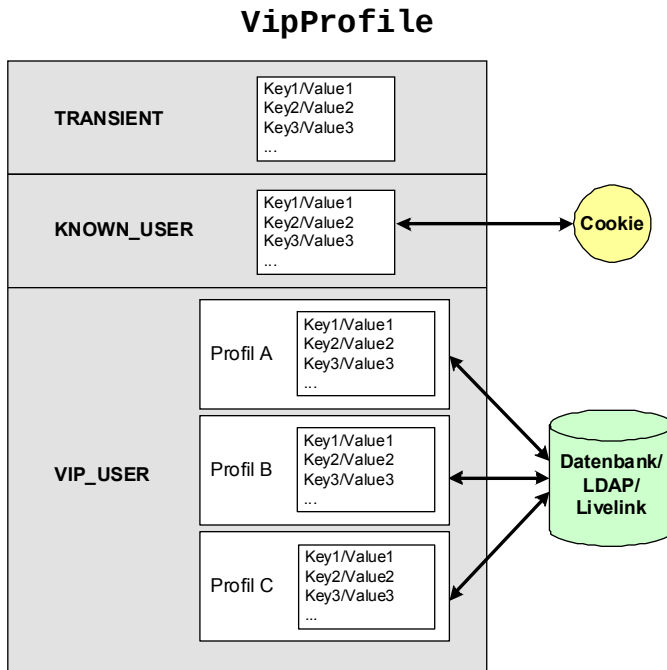


Abb. 33 – Aufbau von VipProfile und der darin verwalteten Daten

Der Zugriff auf einzelne Werte eines Profils und deren Manipulation erfolgt für alle Abschnitte von `VipProfile` über die Methoden `VipProfile.getValue(String attribute, int type)` und `VipProfile.setValue(String attribute, Value value, int type)`. Als `type` ist dabei jeweils der zu benutzende Abschnitt (`TRANSIENT`, `KNOWN_USER` oder `VIP_USER`) anzugeben.

Die Klasse `VipUserBean`

Vollständiger Name: `de.gauss.vip.portalmanager.VipUserBean`

Der logische Name der dieser Klasse zugrunde liegenden Application ist `VipUserApplication`. Die Klasse `VipUserBean` ist eine Schnittstelle zur Benutzerverwaltung von Livelink WCM Server. Sie stellt Methoden zum Speichern und Laden der Value-Objekte verschiedener Repositories sowie Convenience-Methoden für die Behandlung von Bookmark-Einträgen zur Verfügung. Die Klasse `VipUserBean` bietet weiterhin Methoden, die es erlauben, einen individuellen Benutzer darzustellen. Diese Methoden sind deprecated. Stattdessen sollte die Klasse `VipProfile` benutzt werden. Um eine Referenz auf ein Objekt der Klasse `VipProfile` zu erhalten, benutzen Sie die Klasse `SessionBean`.

Die Klasse `VipObjectBean`

Vollständiger Name:

`de.gauss.vip.portalmanager.VipObjectBean`

Die Klasse `VipObjectBean` implementiert den (nur lesenden) Zugriff auf alle Objekte der Websites, die mit Livelink WCM Server verwaltet werden. Insbesondere sind eine Reihe von Methoden enthalten, um den hierarchischen Aufbau der Website zu ermitteln, der durch Themenobjekte strukturiert wird. Dies lässt sich u. a. dazu nutzen, Navigationen aufzubauen.

Das `VipObjectBean` benutzt Repositories der Klasse `VipObjectRepository`, um Daten vom WCM-System abzufragen. Diese Repositories können auf zwei Arten konfiguriert sein:

- durch Angabe des Namens eines Deploymentsystems

In diesem Fall repräsentiert jedes Repository die Daten eines Deploymentsystems. (Für jedes Deploymentssystem wird automatisch ein `VipObjectRepository` bzw.

`VipObjectHandlerRepository` gleichen Namens angelegt; dieser Name kann aber auch so gewählt werden, dass sich der Name des Repository und der des Deploymentsystems unterscheiden.) Ein Deploymentssystem ist zuständig für jeweils genau eine Website und eine Datenhaltungssicht (Edit, QS, Produktion). Mit der Wahl eines `VipObjectRepository` wird also gleichzeitig immer eine bestimmte Website und eine bestimmte Sicht spezifiziert. Der logische Name der zugrunde liegenden Application ist `VipObjectApplication`.

- durch Angabe des Website-Namens und der Datenhaltungssicht auf die Website (Edit, QS, Produktion)

Auf diese Weise kann auf die Objekte einer Website zugegriffen werden, ohne Deploymentssysteme einzurichten. Repositories, die so konfiguriert sind, können keine Deployment-spezifischen Objektdaten (z.B. URL, Pfad) zurückliefern.

Hinweis: Weitere Informationen zur Konfiguration eines `VipObjectRepository` finden Sie in Tabelle 3 "Repositories und ihre Parameter" auf Seite 60.

Einzelne WCM-Objekte sind Einträge in diesen Repositories. Sämtliche Metadaten der WCM-Objekte stellen Schlüssel dieser Einträge dar. Die Bezeichner dieser Metadaten sowie die Werte einiger Metadaten sind im WCM Java API-Programmierhandbuch bzw. in der Javadoc zum Portal Manager API beschrieben.

Die Klasse VipObjectHandlerBean

Vollständiger Name:

`de.gauss.vip.portalmanager.VipObjectHandlerBean`

Das `VipObjectHandlerBean` erweitert das `VipObjectBean` um den schreibenden Zugriff auf die Objekte der mit Livelink WCM Server verwalteten Website. Das `VipObjectHandlerBean` ist die Schnittstelle des Portal Manager API zum WCM Java API.

Die vorgesehenen Methoden sind fast identisch mit dem Methodenumfang des Interface `ObjectHandler` aus dem WCM Java API. Der `ObjectHandler` ist die zentrale Instanz des WCM Java API, um Manipulationen an einer Website über das Programm vorzunehmen. Alle im WCM Java API-Programmierhandbuch beschriebenen Techniken, Einschränkungen, Konstanten usw. gelten somit auch für das `VipObjectHandlerBean`. Kenntnisse der Methoden des `ObjectHandler` sind somit unerlässlich.

Der Name der diesem Bean zugrunde liegenden Application ist `VipObjectApplication`. Die Repositories, die diese Application benutzen, sind in den Konfigurationsdateien **`application.xml`** und **`repository.xml`** eingetragen.

Diese Repositories können auf zwei Arten konfiguriert sein:

- durch Angabe des Namens eines Deploymentsystems

Ein Deploymentsystem ist zuständig für jeweils genau eine Website und eine Datenhaltungssicht (Edit, QS, Produktion). Deshalb bestimmt man durch Wahl eines Deploymentsystems für ein Repository genau, welche Daten zurückgegeben werden und welche Objekte betroffen sind. Mit der Methode `setRepositoryName(String)` können Sie angeben, welches `VipObjectHandlerRepository` von diesem Bean benutzt werden soll, um Operationen im WCM-System auszuführen.

- durch Angabe des Website-Namens und der Datenhaltungssicht auf die Website (Edit, QS, Produktion)

Auf diese Weise kann auf die Objekte einer Website zugegriffen werden, ohne Deploymentsysteme einzurichten. Repositories, die so konfiguriert sind, können keine Deployment-spezifischen Objektdaten (z.B. URL, Pfad) zurückliefern.

Die Datenhaltungssicht, die dem `VipObjectHandlerRepository` indirekt über das Deploymentsystem oder direkt zugeordnet ist, bestimmt, welche Objekte sichtbar sind. Ebenso bestimmt die Datenhaltungssicht, welche Aktionen ausgeführt werden können. Eine detaillierte Beschreibung, welche Aktionen in welcher Sicht ausgeführt werden können, finden Sie in der Beschreibung der Klasse `ObjectHandler` im WCM Java API-Programmierhandbuch.

Für die Benutzung der meisten Methoden des `VipObjectHandlerBean` ist es erforderlich, dass der Benutzer angemeldet ist. Der angemeldete Benutzer muss ausreichende Rechte besitzen, um die Aktionen durchführen zu können. Jeder Benutzer, der Aktionen an WCM-Objekten ausführt, verbraucht eine Lizenz. Diese Lizenz wird wieder freigegeben, wenn sich der Benutzer abmeldet oder die Benutzeranmeldung im Server ungültig wird.

Hinweis: Ein Benutzer, der schreibend auf das WCM-System zugreift, verbraucht eine Lizenz. Diese Lizenz wird frühestens nach einer Minute freigegeben, auch wenn sich der Benutzer vor Ablauf der Minute abmeldet. Nach Ablauf der Minute wird die Lizenz gleichzeitig mit der Abmeldung des Benutzers freigegeben.

Die Klasse `VipObjectFilterBean`

Vollständiger Name:

`de.gauss.vip.portalmanager.VipObjectFilterBean`

Die Klasse `VipObjectFilterBean` ermöglicht, die WCM-Objekte einer Website nach bestimmten Kriterien zu filtern. Alle von der Superklasse `VipObjectBean` geerbten Methoden, die bestimmte Objektmengen liefern, werden gemäß der hier definierten Filterbedingungen zusätzlich eingeschränkt.

Filterergebnisse werden in dem `VipObjectFilterBean` in einem Cache zwischengespeichert. Der Cache wird über den Methodenaufruf `applyFilter` gefüllt. Erneutes Filtern mit identischen Filterbedingungen wird dann nicht tatsächlich durchgeführt, sondern aus dem Cache abgerufen. Der Cache kann explizit gelöscht werden (Methode `invalidateCache`), um eine erneute Filterung zu erzwingen.

Die Klasse `VipWhatsNewBean`

Vollständiger Name:

`de.gauss.vip.portalmanager.VipWhatsNewBean`

Das `VipWhatsNewBean` erweitert die Filterfunktionalität des `VipObjectFilterBean` um das "What's New"-Feature. "What's New" liefert Seiten (WCM-Objekte), die sich in letzter Zeit geändert haben oder neu sind. Dazu wird das Freigabedatum mit einem bestimmten Datum verglichen. Dieses Datum legt fest, wie "in letzter Zeit geändert" bzw. "neu" definiert wird.

Das "What's New"-Feature kann um folgende Eigenschaften erweitert werden:

- Die ermittelte Menge geänderter oder neuer WCM-Objekte kann sortiert werden.
- Einzelne WCM-Objekte können explizit ausgeschlossen werden.

- Objekttypen können explizit ausgeschlossen werden.
- Dateinamenendungen können explizit ausgeschlossen werden.

Von der Superklasse `VipObjectFilterBean` werden folgende Eigenschaften geerbt:

- Neben der "What's New"-Bedingung kann eine weitere Filterbedingung zur Einschränkung angegeben werden.
- Es können WCM-Objekte ab einem bestimmten Wurzelobjekt berücksichtigt werden.
- Die maximale Anzahl von WCM-Objekten kann festgelegt werden.

Es ist möglich, eine Sortierung für die ermittelten Objekte anzugeben. Ohne eine explizite Angabe werden die WCM-Objekte nach ihrem Freigabedatum sortiert, wobei die WCM-Objekte mit dem jüngsten Freigabedatum oben in der Ergebnisliste stehen.

Das Interface `ContentHandler`

Vollständiger Name:

`de.gauss.vip.portalmanager.ContentHandler`

Das Interface `ContentHandler` ermöglicht es, auf den Content eines Objekts, das durch eine URL spezifiziert wird, zuzugreifen.

Der Content kann als String, Stream oder Reader gelesen werden. In allen Fällen kann ein Converter angegeben werden (z.B. `XMLContentConverter`), der den Content konvertiert. Converter interpretieren den Content und geben nur die relevanten Teile zurück.

KAPITEL 5

Hinweise zur Programmierung mit dem Portal Manager API

Dieses Kapitel bietet Ihnen eine knappe Einführung in die Programmierschnittstellen des Portal Manager API und einige Anwendungsbeispiele.

5.1 Authentifizierung und Session-Management

Um die Funktionalität des Portal Manager API nutzen zu können, ist wegen des systemseitigen Zugriffs- und Sicherheitsmechanismus eine ordnungsgemäße Authentifizierung als WCM-Benutzer notwendig. Das für die Kommunikation zwischen Benutzer und dem Portal Manager API verwendete HTTP-Protokoll ist zustandslos. Dies führt dazu, dass für jede HTTP-Anfrage eine Authentifizierung erforderlich ist. Dieser Umstand wird durch die JSP-Spezifikation behoben, die einen Mechanismus für HTTP-Sessions bereitstellt.

Eine HTTP-Session zeichnet sich durch die folgenden Merkmale aus:

- Initialisierung – Eine HTTP-Session wird initialisiert, wenn die Verbindung von einem Client zu einem Webserver das erste Mal hergestellt wird.

- **Session-ID** – Eine Session, die sich im Kontext einer JSP-Engine befindet, wird durch eine für die jeweilige Installation eindeutige Session-ID identifiziert. Die Session-ID wird von der JSP-Engine (Server) verwaltet.
- **HTTP-Session-Timeout** – Eine HTTP-Session ist nur gültig, wenn der Client innerhalb einer bestimmten Zeit eine Anfrage sendet. Sonst tritt aufseiten der JSP-Engine ein Timeout auf.

Der HTTP-Session-Mechanismus wird vom Portal Manager API für das Session-Management der WCM-JavaBeans verwendet und garantiert gleichzeitig die WCM-Authentifizierung für eine bestimmte Session.

Authentifizierung

Die Authentifizierung im Portal Manager API basiert auf dem Konstrukt des WCM-Benutzers, der durch folgende Eigenschaften charakterisiert ist:

- eindeutiger Benutzername (für eine bestimmte Installation oder für den LDAP-Server, der für die Benutzerverwaltung verwendet wird)
- Passwort

Die Benutzerinformation für die Autorisierung im Portal Manager API wird in einer `LoginRepository`-Implementation gespeichert. Es gibt verschiedene `LoginRepository`-Implementationen wie z.B. `VipUserRepository`. Je nach Einsatz des Portal Manager API gibt es verschiedene Authentifizierungsmodi.

- **Authentifizieren über HTTP-Request-Parameter**

Die WCM-Benutzerparameter werden über die HTTP-Request-Parameter `userName` und `userPassword` in der HTTP-Anfrage der JSP-Engine definiert. In diesem Modus werden die Authentifizierungsparameter durch den Einsatz eines HTML-Formulars verfügbar gemacht, das die geforderten Parameter liefert.

- **Authentifizieren über die HTTP-Standardauthentifizierung**

Die WCM-Benutzerparameter werden über den Authentifizierungsmechanismus des (Web-)Servers zur Verfügung gestellt. Der Header der HTTP-Anfrage enthält im Feld *Authorization* den Benutzernamen und das Passwort, die durch einen Doppelpunkt voneinander getrennt und Base 64-kodiert sind.

- **Context-ID-Login aus Request bzw. Cookie**

Die Context-ID wird gelesen und der Benutzer damit angemeldet. Diese ID wird von Secure Access vergeben oder beim Statifizieren in den Request geschrieben.

- **Trusted-Login (Authentifizieren über das Sessionattribut)**

Damit das Portal Manager API mit anderen E-Business-Anwendungen zusammenarbeiten und ein einmaliges Anmelden unterstützen kann, ist es möglich, den Namen des WCM-Benutzers aus einem Attribut abzufragen, das mit der HTTP-Session assoziiert ist. Der Modus Trusted-Login ist eine Funktion des Interface `LoginRepository`. Die folgende Abbildung zeigt, wie das Trusted-Login über den Admin-Client konfiguriert wird.

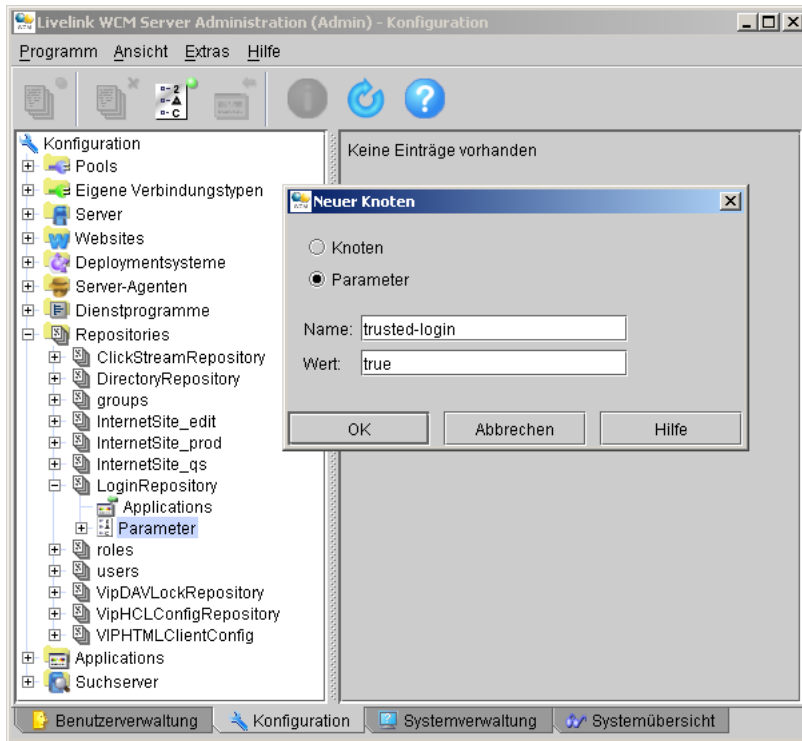


Abb. 34 – Modus Trusted-Login über den Admin-Client konfigurieren

■ Trusted-Login im WCM-System

(`checkLogin(request, "uid", null)`), im Admin-Client muss in den Benutzereinstellungen die Option *Vertraute Anmeldung* aktiviert sein)

Wenn sich der Benutzer nicht am Portal Manager API anmeldet, wird er als anonymer Benutzer mit minimalen Rechten behandelt. Wenn der Modus Trusted-Login bei der Benutzerauthentifizierung verwendet wird, kann die Klasse `DirectoryRepository` nicht verwendet werden, da dieses Repository ein Passwort verlangt, das in diesem Fall nicht zur Verfügung steht.

Wenn auf das WCM-System schreibend zugegriffen werden soll oder die angeforderten Objekte nicht für alle Benutzer lesbar sind, muss sich der Benutzer über das `VipUserRepository` anmelden. Bei dieser Anmeldung wird dem Benutzer eine Context-ID zugeordnet, die das WCM-System für die Rechteüberprüfung benötigt.

Jeder schreibende Benutzer verbraucht eine CA-Lizenz (CA = concurrent author). Diese wird beim Logout bzw. beim Timeout der genutzten Context-ID zurückgegeben.

Hinweis: Die verbrauchte Lizenz wird frühestens nach einer Minute freigegeben, auch wenn sich der Benutzer vor Ablauf der Minute abmeldet. Nach Ablauf der Minute wird die Lizenz gleichzeitig mit der Abmeldung des Benutzers freigegeben.

Session-Management

Im folgenden Klassendiagramm wird das HTTP-Session-Management im Zusammenhang mit der Authentifizierung dargestellt.

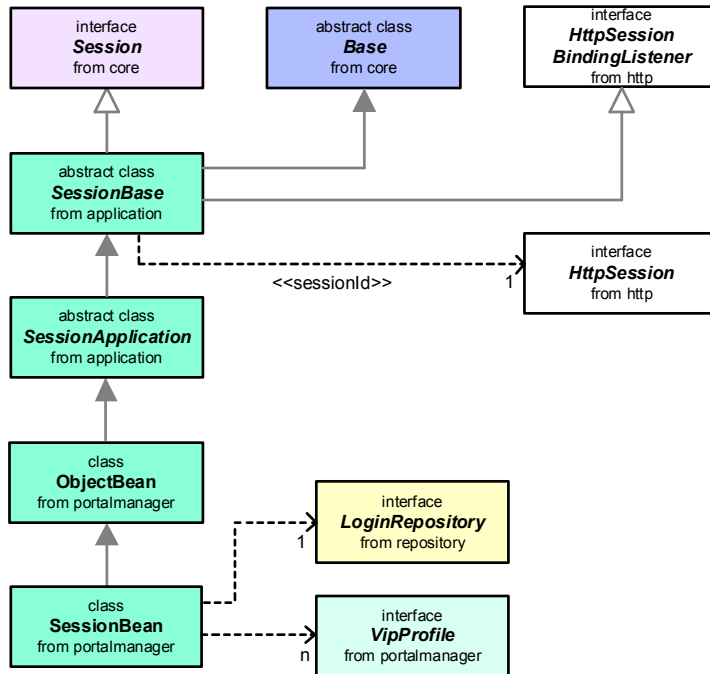


Abb. 35 – Authentifizierung und Session-Management

Für das Session-Management und die Authentifizierung werden die folgenden wesentlichen Komponenten verwendet:

- **HttpSession** – damit kann eine eindeutige Session-ID für eine HTTP-Anfrage ermittelt werden
- **HttpSessionBindingListener** – damit wird der Klasse **SessionBase** angezeigt, dass eine HTTP-Session beendet bzw. gestartet ist
- **LoginRepository** – beinhaltet die Benutzerinformation zur Überprüfung der Anmeldungsanfrage
- **SessionBean** – enthält alle Session-Informationen im Portal Manager API

- `VipProfile` – enthält alle Informationen bezüglich des WCM-Benutzers, der mit einer HTTP-Session assoziiert ist

Login-Repositories

In der folgenden Abbildung sind die Repositories dargestellt, die für eine Anmeldung benutzt werden können.

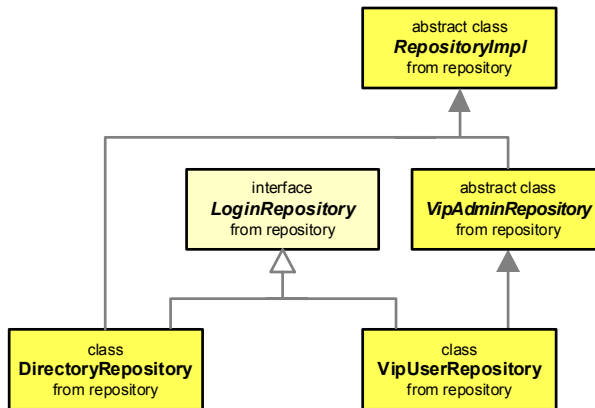


Abb. 36 – Login-Repositories

Manchmal ist es notwendig, eigene Login-Repositories zu schreiben. Wenn Sie z. B. eine Datenbank mit allen Benutzerinformationen haben und die Datenquelle benutzen wollen, um die Authentifizierung durchzuführen, müssen Sie ein Login-Repository entwickeln, das auf die Datenbank zugreift. Diese Klasse muss das Interface `LoginRepository` implementieren und von der abstrakten Basisklasse `RepositoryImpl` oder einer ihrer Unterklassen abgeleitet werden. Weitere Informationen hierzu finden Sie im Abschnitt “Ableiten von der Klasse `RepositoryImpl`” auf Seite 163. Das Standard-Login-Repository ist das `Repository users`.

5.2 Anwendungsbeispiele für das Portal Manager API

In diesem Abschnitt finden Sie zu folgenden Themen Anwendungsbeispiele für die Programmierung mit dem Portal Manager API:

- Anlegen von Benutzern über ein Portal (siehe folgenden Abschnitt)
- Erstellen dynamischer Navigationselemente (siehe Abschnitt “Dynamische Navigationselemente erzeugen” auf Seite 150)
- Multicontent-Seiten (siehe Abschnitt “Mehrere Content-Objekte auf einer Seite (Multicontent-Seite) platzieren” auf Seite 154)
- Formularbasierte Objekterstellung (siehe Abschnitt “WCM-Objekte über ein HTML-Formular erstellen” auf Seite 158)

Benutzer über ein Portal in einem LDAP-System anlegen

Die Nutzung von Internetportalen erfordert in der Regel, dass ein Benutzer seine Identität angibt. Dazu richtet sich der Benutzer im Portal ein Benutzerkonto mit seinen persönlichen Daten ein. Über dieses Konto kann der Benutzer das Aussehen des Portals den eigenen Wünschen und Bedürfnissen anpassen. Außerdem lässt sich steuern, auf welche vom Portal bereitgestellten Inhalte der Benutzer zugreifen kann.

Die Benutzerdaten können in einer Datenbank oder in einem LDAP-Verzeichnisdienst gespeichert werden. In diesem Abschnitt wird beschrieben, wie sich mithilfe des `DirectoryRepository` aus dem Portal Manager API neue Benutzerkonten über ein Portal in einem LDAP-Verzeichnisdienst anlegen lassen. Die dazu benötigten Daten können aus Anmeldeformularen übernommen werden. So können Portalbenutzer ihre Konten selbst anlegen und verwalten.

Es lassen sich auch JSP-Applikationen zur Verwaltung von Benutzerkonten entwickeln. Derartige Web-Applikationen bieten sogar potenziell mehr Funktionalität als der Admin-Client, da sich in ihnen beliebige Daten speichern und verwalten lassen.

Voraussetzungen

- Damit ein Benutzer, dessen Konto im LDAP-Verzeichnisdienst angelegt wird, sich am WCM-System anmelden kann und Teil der Zugriffssteuerung wird, muss die WCM-Benutzerverwaltung auf LDAP basieren. Die neu angelegten Benutzerkonten müssen von dem Administrationsserver (Master oder Proxy) aus erreichbar sein, der für den Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft, den Anmeldevorgang durchführt.
- Für das `DirectoryRepository` sind alle LDAP-Verzeichnisdienste nutzbar, die auch für die WCM-Benutzerverwaltung verwendet werden können (genaue Angaben zu den unterstützten LDAP-Verzeichnisdiensten enthält die Freigabemitteilung von Livelink WCM Server).

`DirectoryRepository` konfigurieren

Um auf einen LDAP-Verzeichnisdienst zugreifen zu können, muss ein Repository vom Typ `DirectoryRepository` konfiguriert werden. Gehen Sie dazu folgendermaßen vor:

1. Legen Sie im Admin-Client ein neues Repository an (siehe Abschnitt "Repository anlegen" auf Seite 56).

Geben Sie dabei als Klassenname `de.gauss.vip.portalmanager.repository.DirectoryRepository` an.
2. Konfigurieren Sie das neue Repository über Parameter (siehe Abschnitt "Parameter eines Repository festlegen" auf Seite 58).

Die folgende Tabelle enthält eine Beispielkonfiguration. Grundlegende Informationen zum DirectoryRepository erhalten Sie im Abschnitt “Die Klasse DirectoryRepository” auf Seite 91. Eine Beschreibung der einzelnen Parameter finden Sie in Tabelle 3 “Repositories und ihre Parameter” auf Seite 60.

Hinweise:

Die Werte der ldapattr-Parameter müssen gegebenenfalls an die Konfiguration des LDAP-Verzeichnisdienstes angepasst werden.

Es müssen nur Mappings für die Parameter vorgenommen werden, bei denen sich die Werte von pomaattr- und ldapattr-Parameter unterscheiden. Dabei wird zwischen Groß- und Kleinschreibung unterschieden, d. h., die Werte “userpassword” und “userPassword” sind **nicht** identisch.

Tabelle 6 – Beispielkonfiguration eines DirectoryRepository

Parameter	Wert
url	ldap://ldapserver.company.example:389
authentication	simple
principal	{LDAP-Benutzer mit dem Recht, Einträge hinzuzufügen}
credentials	{Passwort des unter principal eingetragenen Benutzers}
principal-changes	{LDAP-Benutzer mit dem Recht, Einträge zu ändern und zu löschen} Hinweis: Diese Parameter wird dazu verwendet, Benutzer zu Gruppen und Rollen zuzuordnen. Dies gilt auch für den Fall, dass der angemeldete Benutzer nicht das Recht hat, LDAP-Einträge zu ändern und zu löschen.

Parameter	Wert
credentials-changes	{Passwort des unter principal-changes eingetragenen Benutzers}
search-scope	SUBTREE_SCOPE
base	ou=software, o=company.example
group-base	ou=groups, ou=software, o=company.example
role-base	ou=roles, ou=software, o=company.example
dn-pattern	uid=*, ou=software, o=company.example
password-attr	userPassword
[- mappings	
[- mapping	
[- LDAP_USER_ID	
pomaattr	uid
ldapattr	uid
[- CN	
pomaattr	cn
ldapattr	cn
[- INIT_PASSWORD	
pomaattr	pwdChange
ldapattr	initPassword
[- VIP_ACCESS	
pomaattr	account

Parameter	Wert
ldapattr	vipAccess
☐ LDAP_OBJECTCLASS	
pomaattr	objectclass
ldapattr	objectClass
☐ TRUSTED_LOGIN	
pomaattr	trustedLogin
ldapattr	trustedLogin
☐ VIP_FUNCAREAS	
pomaattr	vipFuncarea
ldapattr	vipFuncarea
☐ VIP_RIGHTS	
pomaattr	vipRights
ldapattr	vipRights
☐ LANGUAGE	
pomaattr	language
ldapattr	preferredLocale
☐ LDAP_VIP_TYPE	
pomaattr	vipType
ldapattr	vipType
☐ VIP_WEBSITES	
pomaattr	website

Parameter	Wert
ldapattr	vipWebsite
☐ HCL_PROFILES	
pomaattr	hclProfiles
ldapattr	hclProfiles
☐ MAIL	
pomaattr	email
ldapattr	mail
☐ VIP_MEMBERS	
pomaattr	user-name
ldapattr	member
☐ VIP_SUBSTITUTE	
pomaattr	vipSubstitute
ldapattr	vipSubstitute
☐ USER_PASSWORD	
pomaattr	userpassword
ldapattr	userPassword

3. Ordnen Sie der Application VipUserApplication das neue Repository zu (siehe Abschnitt "Repository zu einer Application zuordnen" auf Seite 86).

Methoden zur Benutzung des DirectoryRepository

Um auf die Daten des LDAP-Verzeichnisdienstes zugreifen und diese beeinflussen zu können, stehen die Methoden zur Verfügung, die die Klasse `VipUserBean` von der Klasse `ObjectBean` erbt:

- `containsKey("<repositoryName>", key)`
- `createEntry("<repositoryName>", key, data)`
- `loadEntry("<repositoryName>", key)`
- `removeEntry("<repositoryName>", key)`
- `storeEntry("<repositoryName>", key, data)`

Der Parameter `key` dieser Methoden muss ein `String` oder ein `StringValue` sein. Der Wert des Parameters muss der voll qualifizierte "distinguished name" (DN), ein Benutzer-, Gruppen- oder Rollename sein. Wenn kein DN als Parameter `key` spezifiziert wurde, sucht das `DirectoryRepository` in der folgenden Reihenfolge nach einem Principal:

1. nach einem Benutzer, dessen DN aus dem Wert des Parameters `dn-pattern` des Repository gebildet ist
2. nach einer Gruppe mit dem ermittelten DN `cn={key}, {group-base}`
3. nach einer Rolle mit dem ermittelten DN `cn={key}, {role-base}`

Der Parameter `data` der Methoden wie auch die Rückgabewerte der meisten Methoden sind `RepositoryEntry`-Objekte. Diese `RepositoryEntry`-Objekte enthalten die Attribute des LDAP-Eintrags, der durch `key` referenziert wird. Diese Attribute werden – entsprechend dem in der Konfiguration des Repository festgelegten Mapping – konvertiert, wenn sie in den LDAP-Verzeichnisdienst geschrieben oder von dort gelesen werden.

Die Methoden `containsKey`, `createEntry` und `loadEntry` werden im Kontext des Repository-Bindeprofils, das durch die Parameter `principal` und `credential` spezifiziert ist, ausgeführt. Deshalb lassen sich diese Methoden unabhängig vom aktuell angemeldeten Benutzer ausführen. Vorausgesetzt wird dabei, dass der Bindeprofil-Benutzer über ausreichende Rechte im LDAP-Verzeichnisdienst verfügt.

Die Methoden `removeEntry` und `storeEntry` werden im Kontext des aktuell angemeldeten Benutzers ausgeführt, sofern die Parameter `principal-changes` und `credential-changes` für das Repository konfiguriert wurden.

Die Methode `createNewUser("{Repository-Name}", key, data)` der Klasse `VipUserBean` ermöglicht es, in einem Schritt neue Benutzer anzulegen und zu Gruppen oder Rollen zuzuordnen. Nachdem ein Benutzer einer Gruppe oder Rolle zugeordnet wurde, müssen die Daten der Gruppe oder Rolle aktualisiert werden. Deshalb muss entweder der aktuell angemeldete Benutzer über ausreichende Rechte für diese Operation verfügen oder müssen die Parameter `principal-changes` und `credential-changes` zusammen verwendet werden.

Beispiel für das Anlegen eines neuen Benutzerkontos

In diesem JSP-Beispiel werden die Eingaben aus einem HTML-Formular dazu benutzt, ein neues Benutzerkonto anzulegen.

```
<html>
<head></head>
<body>
<%@ page
    import="java.util.*,
           de.gauss.lang.*,
           de.gauss.vip.util.*,
           de.gauss.vip.repository.*"
%>
<jsp:useBean id="vub"
             class="de.gauss.vip.portalmanager.VipUserBean"
             scope="session"/>

<%
    // read the parameters from the request
```

```
String name = request.getParameter("name");
if (name == null) name = "";
String uid = request.getParameter("uid");
if (uid == null) uid = "";
String mail = request.getParameter("mail");
if (mail == null) mail = "";
String pwd = request.getParameter("pwd");
if (pwd == null) pwd = "";

%>
<p><form action="{VIPURL}">
    <table>
        <tr><td>Name</td><td><input name="name"
                                value="<%=name%>"></td></tr>
        <tr><td>User-Id</td><td><input name="uid"
                                value="<%=uid%>"></td></tr>
        <tr><td>E-Mail</td><td><input name="mail"
                                value="<%=mail%>"></td></tr>
        <tr><td>Password</td><td><input name="pwd" type="password"
                                value="<%=pwd%>"></td></tr>
        <tr><td>&nbsp;</td><td><input type="submit"
                                value="create user"></td></tr>
    </table>
</form></p>
<hr>
<%
    if ((name.length() > 0) && (uid.length() > 0) && (pwd.length()
        > 0))
    {
        RepositoryEntry newUserEntry = new RepositoryEntry();
        // add the common name
        newUserEntry.putValue(AttributeName.CN,
                                new StringValue(name));
        // add the user id
        newUserEntry.putValue(AttributeName.UID,
                                new StringValue(uid));
        // add the mail address
        newUserEntry.putValue(AttributeName.MAIL,
                                new StringValue(mail));
        // add the password
        newUserEntry.putValue(AttributeName.PWD,
                                new StringValue(pwd));
        // allow WCM access
        newUserEntry.putValue(AttributeName.ACCOUNT, new
                                BooleanValue(true));
        // define a language
        newUserEntry.putValue(AttributeName.LANGUAGE,
                                new LocaleValue(Locale.getDefault()));
    }
}
```

```
// the user does not have to change his password
newUserEntry.putValue(AttributeName.PWD_CHANGE,
                      new BooleanValue(false));

// no trusted login allowed
newUserEntry.putValue("trustedLogin",
                      new BooleanValue(false));

// no default object rights are defined
newUserEntry.putValue("vipRights", new StringValue(" "));
// the user is assigned two object classes: vip and vipUser
RepositoryMap objectclassMap = new RepositoryMap();
objectclassMap.putEntry("vip", null);
objectclassMap.putEntry("vipUser", null);
newUserEntry.putValue(AttributeName.OBJECTCLASS,
                      objectclassMap);

// assign the user to the website
RepositoryMap websiteMap = new RepositoryMap();
websiteMap.putEntry("{VIPSITE}", null);
newUserEntry.putValue(AttributeName.WEBSITES, websiteMap);
// assign the user to a group
RepositoryMap groupMap = new RepositoryMap();
groupMap.putEntry("testgroup", null);
newUserEntry.putValue(AttributeName.GROUPS, groupMap);
// create the user
vub.createNewUser("ldap", uid, newUserEntry);

%><p>Created new user account: <%=vub.loadEntry("ldap",
                      uid)%></p><%
}
else
{
    %><p>Please insert data into all fields.</p><%
}
%>
</html>
```

Dynamische Navigationselemente erzeugen

Die konsistente Navigation einer Website dient dazu, den Content für den Benutzer auf einfache Weise zugänglich zu machen. Darüber hinaus ermöglicht die Navigation einem Benutzer durch Hyperlinks einen schnellen, unkomplizierten Wechsel zu assoziiertem Content, etwa zu themenverwandten Seiten oder zu Seiten, die in der hierarchischen Struktur benachbart sind. Auch Sitemaps gehören zu den Navigationselementen. Sie bereiten jedoch im Allgemeinen nicht den Kontext der aktuellen Seite auf, sondern Informationen über die gesamte Website.

In Livelink WCM Server stehen allein durch die Speicherung von Metadaten zu jedem Content-Objekt schon genügend Informationen für eine Navigation zur Verfügung. So werden zu jedem Content-Objekt diejenigen Objekte gespeichert, die per Hyperlink erreichbar sind, sowie die Objekte, von denen aus das aktuelle Content-Objekt per Hyperlink erreichbar ist. Außerdem werden die Verweise auf das in der Themenstruktur übergeordnete sowie auf alle untergeordneten bzw. auf gleicher Ebene liegenden WCM-Objekte gespeichert. Weitere Informationen für eine Navigation sind durch den Themenbaum und die eventuelle Verknüpfung eines Content-Objekts mit einem Vorlageobjekt gegeben.

Wenn sich die zugrunde liegenden Metadaten ändern, ändert sich – bei einer entsprechenden Implementation – der Inhalt automatisch, ohne dass manuelle Anpassungen nötig sind. Löschen, Kopieren, Verschieben von Objekten führt dann nicht mehr dazu, dass die Navigationselemente veralten.

Mit dem Portal Manager API lassen sich Navigationselemente dynamisch erzeugen. So können etwa die Unterthemen eines Hauptthemas automatisiert als Unterpunkte eines Menüs ausgegeben werden oder Sitemaps dem Benutzer einen Überblick über die Website-Struktur geben. In beiden Fällen werden Änderungen in der Struktur der Website ohne erneute Bearbeitung für die Navigation berücksichtigt.

Werden Navigationselemente mit Methoden des Portal Manager API dynamisch generiert, so können die Zugriffsrechte der Benutzer berücksichtigt werden. Ein Benutzer hat nur die Navigationsmöglichkeiten, für die er berechtigt ist. So sieht beispielsweise ein anonymer Benutzer andere Elemente als ein autorisierter Benutzer. Weitergehende Konzepte der Personalisierung würden Objektinformationen auf Basis von Merkmalen des Benutzerprofils filtern.

Konzeption

In diesem Abschnitt wird skizziert, wie Sie dynamische Navigationselemente konzipieren. Gehen Sie dazu folgendermaßen vor:

1. Skizzieren Sie das Layout der Seite und überlegen Sie sich, mit welchen HTML-Elementen sich Ihr Layout umsetzen lässt.
2. Schreiben Sie auf Basis dieses Konzepts den JSP-Code. Überlegen Sie sich hierbei, welche Teile des HTML-Codes dynamisch erzeugt werden müssen und welche Beans die passenden Methoden bereitstellen, um an diese Informationen zu gelangen.

Normalerweise werden Hyperlinks auf WCM-Objekte unterhalb eines bestimmten Themas als Tabelle oder Liste angeboten. Anzahl und Namen der Objekte sind unbekannt, d.h., es werden die Namen bzw. die Überschriften sowie die URLs aller WCM-Objekte unterhalb des gewählten Themas benötigt.

Mithilfe der Klasse `VipObjectBean` kann ein `RepositoryIterator` auf alle Objekte unterhalb des Themas geliefert werden. Dabei ist es auch möglich, nur Objekte eines bestimmten Typs zu berücksichtigen.

Über den `RepositoryIterator` wird die Liste von Objekten durchlaufen. Jeder Eintrag ist ein `RepositoryEntry`, der Metadaten (Name, Überschrift usw.) enthält, die ausgelesen und in HTML ausgegeben werden können.

Es ist auch möglich, sich die URL eines Objekts über das `RepositoryEntry` ausgeben zu lassen, wobei jedoch immer nur die URL aus dem entsprechenden Deploymentsystem angegeben werden kann. Sinnvollerweise sollten jedoch jeweils die Hyperlinks des Systems angezeigt werden, in dem sich der Benutzer gerade befindet. Eine entsprechende Methode wird von der Klasse `VipObjectBean` bereitgestellt.

Realisierung

Im Folgenden wird das Aufbauen der JSP-Seite skizziert.

1. Importieren Sie die benötigten Klassen. Beans instanziiieren Sie mit einer speziellen JSP-Anweisung.
2. Stellen Sie sicher, dass alle verwendeten WCM-Beans initialisiert sind.

Der Klasse `VipObjectBean` muss bekannt sein, welches Repository verwendet werden soll. Das Repository legt das Deploymentsystem fest. Für das Repository wird üblicherweise das WCM-Tag `{VIPDEPLOYMENT_NAME}` genutzt, das auf das Deploymentsystem des Objekts verweist. Das WCM-Tag `{VIPOID}` verweist hingegen auf die OID des WCM-Objekts.

Das folgende Code-Beispiel enthält den JSP-Code, mit dem der `RepositoryIterator` gelesen und durchlaufen wird (betrifft alle JSP-Themen unterhalb der WCM-OID). Die Anweisungen werden so in den HTML-Code kodiert, dass z.B. für jeden `RepositoryEntry` ein Listeneintrag oder eine Tabellenzeile ausgegeben wird. Ein Hyperlink kann dann aus dem Namen des Eintrags und der URL des WCM-Objekts zusammengesetzt werden.

```
<html>

<head>
<TITLE>getSubElements</TITLE>
</head>

<body>

<!-- This should be part of the Template -- Begin -->
<%@ page
    import="de.gauss.vip.repository.RepositoryEntry,
           de.gauss.vip.repository.RepositoryIterator,
           de.gauss.lang.KeySort,
           de.gauss.lang.StringValue"
%>

<jsp:useBean id="sessionBean"
class="de.gauss.vip.portalmanager.SessionBean"    scope="session" />
<jsp:useBean id="objBean"
class="de.gauss.vip.portalmanager.VipObjectBean"  scope="session" />

<%
    objBean.setRepositoryName("{VIPDEPLOYMENT_NAME}");

    // A user has to be logged in, otherwise the operations are
    // executed in the content of "Anonymous".
    if( !sessionBean.isLoggedIn(request) )
    // The request contains the necessary attributes for the
    // login.
        sessionBean.checkLogin( request );
%>
<!-- This should be part of the Template -- End -->
<%
    // get the meta data of the object
    RepositoryEntry entry = objBean.getEntry( "{VIPOID}" );
    if( entry != null )
    {
        KeySort sort = new KeySort(new StringValue("title"));
        // get all subelements of the current object which are
        // jsp topics
        RepositoryIterator it = objBean.getSubElements( entry,
            "JSPTOPIC", sort);
        %><table><%
        while( it.hasNext() )
        {
            RepositoryEntry e = it.nextEntry();
            %>
            <tr><td>
```

```
        <a href="<%= e.getValue("url") %>">
        <%=e.getValue("title") %>
        </a>
    </td></tr>
    <%
    }
    %></table><%
    }
%>

</body>

</html>
```

Mehrere Content-Objekte auf einer Seite (Multicontent-Seite) platzieren

Mithilfe des Portal Manager API können beliebig viele Content-Objekte (Texte, Tabellen, Grafiken usw.) auf einer Seite präsentiert werden. Dabei müssen einschränkende Aspekte wie z.B. Zugriffsrechte natürlich berücksichtigt werden. Als Content können Objekte aus der WCM-verwalteten Website ebenso benutzt werden wie Daten aus externen Anwendungen. Sogar website- und anwendungsübergreifende Content-Verknüpfungen sind völlig unproblematisch.

Der Ansatz der Multicontent-Seite ist besonders geeignet für anspruchsvolle Websites, bei denen in der Regel unterschiedliche Mitarbeiter für das Erstellen bzw. die Bereitstellung des Content zuständig sind. So werden z.B. Artikelbeschreibungen von Werbetextern erstellt, die Abbildungen liefert eine Bildagentur, und Artikelstammdaten wie Name, Artikelnummer, Preis und Lieferfrist werden von Mitarbeitern aus dem Einkauf in einer Datenbank abgelegt.

Konzeption

Eine Multicontent-Seite wird als JSP-Seite mit den Methoden des Portal Manager API erstellt. Dies erfordert bereits an dieser Stelle eine Arbeitsteilung zwischen Webdesigner und Programmierer.

Skizzieren Sie das Layout der Multicontent-Seite auf einem Blatt Papier. Da es sich empfiehlt, die Content-Objekte mithilfe von HTML-Tabellen anzuordnen, sollten Sie Position und Größe der Tabellen bereits jetzt genau festlegen.

Realisierung

Wenn Sie mit der Erstellung des Layouts fertig sind, gehen Sie folgendermaßen vor.

1. Legen Sie ein JSP-Objekt an.
2. Setzen Sie Ihren Entwurf mithilfe von HTML-Elementen um und kontrollieren Sie das Ergebnis im Browser. Nehmen Sie gegebenenfalls Änderungen vor, bis das Ergebnis Ihren Vorstellungen entspricht. Dazu müssen Sie die Seite möglicherweise mehrmals ausleihen und zurückgeben.
3. Fügen Sie den JSP-Code in die Seite ein. Dabei sollten Sie darauf achten, die JSP-Elemente so weit wie möglich vom HTML-Code zu trennen. Dies erhöht die Übersichtlichkeit einer JSP-Seite entscheidend und ermöglicht es Ihnen so, später erforderliche Änderungen leichter umzusetzen.

Das folgende Code-Beispiel enthält den JSP-Code für eine Multicontent-Seite, in der die verschiedenen Möglichkeiten beim Laden des Contents angezeigt werden.

```
<html>

<head>
<TITLE>multiContent</TITLE>
<body>

<!-- This should be part of the Template -- Begin -->
<%@ page
    import="de.gauss.vip.repository.RepositoryEntry,
           de.gauss.vip.portalmanager.converter.HTMLContentConverter,
           de.gauss.vip.portalmanager.ContentHandler"
%>
<jsp:useBean id="sessionBean"
    class="de.gauss.vip.portalmanager.SessionBean"
    scope="session" />
<jsp:useBean id="objBean"
    class="de.gauss.vip.portalmanager.VipObjectBean"
    scope="session" />

<%
    objBean.setRepositoryName("{VIPDEPLOYMENT_NAME}");

    // A user has to be logged in, otherwise the operations are
    // executed in the content of "Anonymous".
    if( !sessionBean.isLoggedIn( request ) )
    // The request contains the necessary attributes for the
    // login.
        sessionBean.checkLogin( request );
%>
<!-- This should be part of the Template -- End -->
<%

    // In this example is no check, if the retrieved entries
    // are null in my website is object "215" a jsp-object
    RepositoryEntry jsp  = objBean.getEntry("215");
    ContentHandler chJsp = objBean.getContent(jsp);

    // in my website is object "255" a picture object
    RepositoryEntry pic  = objBean.getEntry("255");
    ContentHandler chPic = objBean.getContent(pic);
%>
<table>
<tr>
    <td><!-- Gets the whole content, i. e. content plus
        template. -->
        <%= chJsp.getString() %>
    </td>
```

```
<td><!-- Returns the content, in this case the <img> tag -->
    <%= chPic.getString(new HTMLContentConverter()) %>
</td>
</tr>
<tr>
    <td><!-- This is faster, because it is not necessary to load
        the content -->
        ">
    </td>
    <td><!-- Gets only the content of the object, i. e. content
        without template. -->
        <%= objBean.getVipContent(jsp) %>
    </td>
</tr>
</table>

</body>
</html>
```

Mit der Methode `getVipContent` der Klasse `VipObjectBean` können Sie auf den HTML-Code der Content-Objekte zugreifen. Darüber hinaus stellt dieses Bean die Methode `getContent` bereit, die einen `ContentHandler` zurückliefert. Über diese Klasse kann auf bestimmte Inhalte im Content zugegriffen werden (z. B. auf bestimmte Attribute in Formularinstanzen). Der Content wird über eine URL-Verbindung geladen, d. h. dynamische Seiten werden ausgeführt, bevor sie zurückgeliefert werden. Diese Methoden erfordern als Parameter ein `RepositoryEntry`. Das Portal Manager API stellt Ihnen zahlreiche Methoden zur Verfügung, um ein `RepositoryEntry` indirekt zu ermitteln. Wenn Ihnen die Objekt-ID eines Content-Objekts bekannt ist, können Sie ein `RepositoryEntry` jedoch direkt mit der Methode `getEntry` der Klasse `VipObjectBean` ermitteln.

WCM-Objekte über ein HTML-Formular erstellen

Mit dem Portal Manager API lassen sich HTML-Formulare entwickeln, mit denen Content formularbasiert erstellt werden kann. Eine formularbasierte Eingabe ist immer dann vorteilhaft, wenn viele gleichartige Content-Objekte erstellt werden sollen und die zuständigen Benutzer eine vereinfachte Benutzeroberfläche für die Umsetzung benötigen.

Obwohl der Content-Client sehr einfach zu bedienen ist, stellt er doch einen erheblichen Funktionsumfang bereit. Für die Erstellung einfacher, identisch strukturierter Dokumente wie Artikelseiten und Pressemitteilungen wird jedoch nur ein kleiner Teil dieser Funktionen benötigt. Durch Bereitstellung dieser Funktionen über ein Formular lässt sich die Einarbeitungszeit der Benutzer beträchtlich verkürzen.

Letzteres ist unter anderem dann wichtig, wenn externe Ressourcen in den Erstellungsprozess einer Website eingebunden werden sollen. Ein professionelles Web-Content-Management-System wie Livelink WCM Server erlaubt es, dass externe Mitarbeiter dezentral an ihrem jeweiligen Standort arbeiten. So können etwa die Werbetexter und Bildagenturen zur Erstellung von Artikelseiten auf diese Weise eingebunden werden.

Ein Formular gibt die Struktur des damit erzeugten Content-Objekts fest vor. Erzeugt werden XML-Objekte – so genannte *Formularinstanzen* –, deren Struktur der des Formulars entspricht. Die Elemente dieser XML-Objekte können dann auf einer HTML-Seite beliebig angeordnet und formatiert werden.

Hinweis: Es können natürlich auch andere Objekttypen erzeugt werden.

Konzeption

Die formularbasierte Eingabe muss immer als klar definierte Anwendung realisiert werden. Die Benutzeroberfläche besteht aus mehreren Komponenten, die in einen Workflow eingebunden werden und auf die jeweilige Anwendung abgestimmt sind. In der Regel werden folgende Komponenten benötigt:

- Übersichtsseite – listet die erstellten Formularinstanzen auf und stellt Schaltflächen für die an diesen WCM-Objekten ausführbaren Aktionen bereit
- Eingabeformular – dient zum Anlegen von Formularinstanzen und stellt Eingabefelder, Auswahlfelder etc. zur Erstellung der XML-Objekte bereit
- Änderungsformular – dient zur Änderung bestehender Formularinstanzen. Die Feldinhalte bzw. der Zustand der Formularinstanzen werden editierbar dargestellt.
- Ansichtsseite – dient zur Aufbereitung der Feldinhalte und des Zustands der Formularinstanzen für die Darstellung
- `VipObjectHandlerBean` – führt u. a. die folgenden Aktionen auf den Formularinstanzen aus:
 - Anlegen
 - Ändern
 - Vorlegen
 - Ablehnen
 - Freigeben
 - Löschen

Das Eingabeformular wird von der Übersichtsseite aus aufgerufen. Der Benutzer gibt die Formulardaten ein und schickt das Formular ab. Der Inhalt des Formulars wird an die JSP-Seite zum Anlegen übergeben.

Diese extrahiert die Formulareingaben aus der HTTP-Anfrage und erzeugt daraus die Metadaten und den Content für die Formularinstanz. Die Formularinstanz muss anschließend als WCM-Objekt angelegt und ihr Content geschrieben werden.

Soll eine bestehende Formularinstanz geändert werden, wird sie von der Übersichtsseite aus in das Änderungsformular geladen. Der Benutzer ändert die Daten und schickt erneut das Formular ab. Abweichend zum Anlegen von Formularinstanzen wird der Formularinhalt jetzt an die JSP-Seite zum Ändern gegeben. Diese extrahiert die Formulardaten und aktualisiert damit Metadaten und Content der zu ändernden Formularinstanz.

Die übrigen Aktionen, die von der Übersichtsseite aus durchführbar sind, sind weniger spektakulär. Es wird lediglich der entsprechende `ObjectHandler` für die Formularinstanz aufgerufen und die Übersichtsseite aktualisiert.

5.3 Framework für Applications

In diesem Abschnitt wird das Framework beschrieben, das für die Realisierung von benutzerspezifischen Applications verwendet wird.

Das Portal Manager API stellt eine Reihe von Java-Klassen zur Verfügung, mit denen benutzerspezifische Applications in das Portal Manager API integriert werden können. Das Zusammenspiel der Klassen wird im folgenden Klassendiagramm abgebildet:

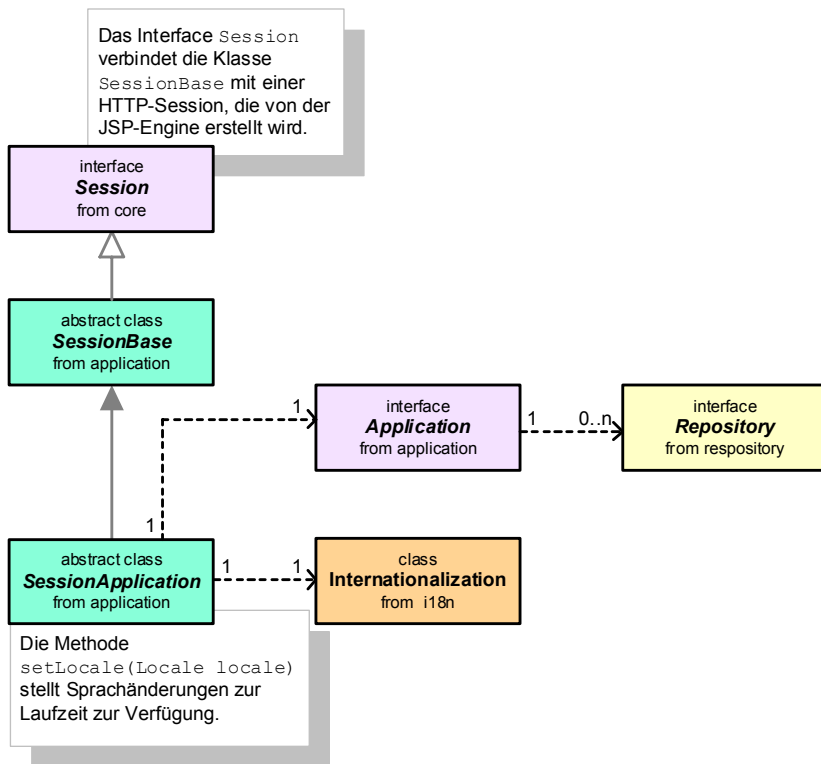


Abb. 37 – Klassendiagramm für Applications

Folgende Elemente sind zur Realisierung einer Application notwendig:

- **Repository** – Eine Implementation des Interface `Repository` wird als Wrapper-Klasse genutzt, um eine externe Datenquelle WCM-fähig zu machen. Eine Application kann mit 0–n Repositories verbunden werden.
- **Application** – Eine Implementation des Interface `Application` wird als Wrapper-Klasse verwendet, um die Logik einer Application WCM-fähig zu machen. Die Application verwendet die assoziierten Repositories, um die Business-Objekte zu erhalten und zu pflegen.

- **JavaBean** – Eine von der Klasse `SessionApplication` ableitende Klasse packt die `Application` in ein `JavaBean`, das die Instanziierung der `Application` im Kontext einer JSP-Seite vornimmt. Das Bean stellt die folgende Funktionalität zur Verfügung:
 - Verbinden des Bean mit einer `javax.servlet.http.HttpSession` – Die HTTP-Session stellt eine Zustandsunterstützung für das zustandslose HTTP-Protokoll zur Verfügung. Dadurch wird es möglich, einen bestimmten Benutzer mit einer HTTP-Session für die Authentifizierung zu assoziieren.
 - zur Laufzeit umschaltbare Internationalisierungsunterstützung für eine `Application` über die Klasse `Internationalization`
 - zusätzliche mit der `Application` in Beziehung stehende Funktionalität

Implementieren einer Application

Der eigentliche Vorgang des Implementierens einer `Application`, die vom Portal Manager API abgeleitet ist, umfasst folgende Schritte:

1. Implementieren der Repositories, `Applications`, `SessionApplication`-Objekte
2. Konfiguration des Portal Manager API, um die neue `Application` zu verwenden (siehe Abschnitt 3.2 “Applications verwalten” ab Seite 69)

Ableiten der Application-spezifischen Klassen

Um dem Benutzer den Arbeitsablauf zu erleichtern, stellt das Portal Manager API abstrakte Basisklassen mit den Basisimplementationen aller Funktionalitäten zur Verfügung, die mit dem Portal Manager API zusammenhängen. Eine benutzer-spezifische `Application` muss dann nur eine bestimmte Basisklasse implementieren. Die

allgemeine Hierarchie zum Entwickeln von Applications ist in der folgenden Abbildung dargestellt:

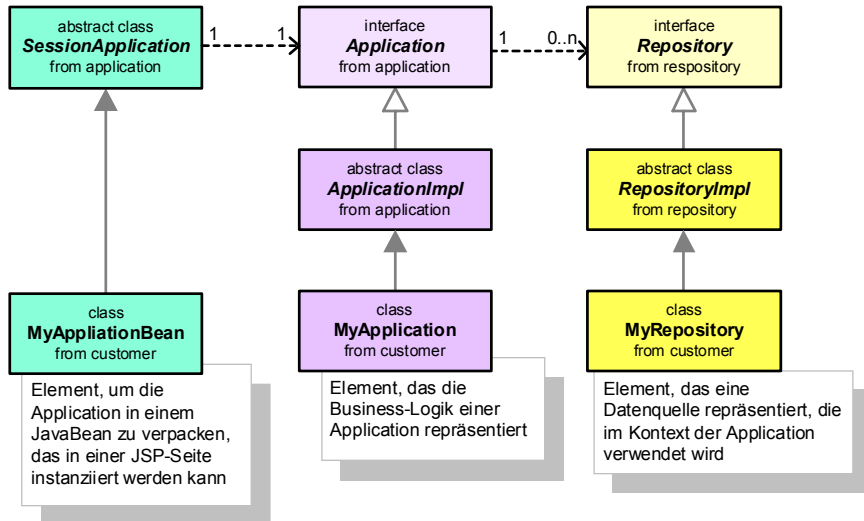


Abb. 38 – Allgemeine Hierarchie zum Entwickeln von Applications

Ableiten von der Klasse RepositoryImpl

Wie bereits erwähnt, fungiert das Interface Repository als Wrapper-Klasse, um Datenquellen so aufzubereiten, dass das Portal Manager API auf sie zugreifen kann.

Die abstrakte Basisklasse RepositoryImpl stellt Standardimplementationen für Methoden zur Verfügung, die mit dem Portal Manager API in Bezug stehen. Es müssen deshalb nur die Methoden, die für die Datenquellen relevant sind, in den abgeleiteten Repository-Klassen implementiert werden. Ein Repository im Kontext des Portal Manager API basiert auf den Datenstrukturen, die innerhalb des WCM-Systems verwendet werden (siehe Abschnitt "RepositoryMap und zugehörige Klassen" auf Seite 107).

In der folgenden Abbildung werden alle auf Datenquellen bezogenen Methoden dargestellt, die von der Klasse `RepositoryImpl` zur Verfügung gestellt werden.

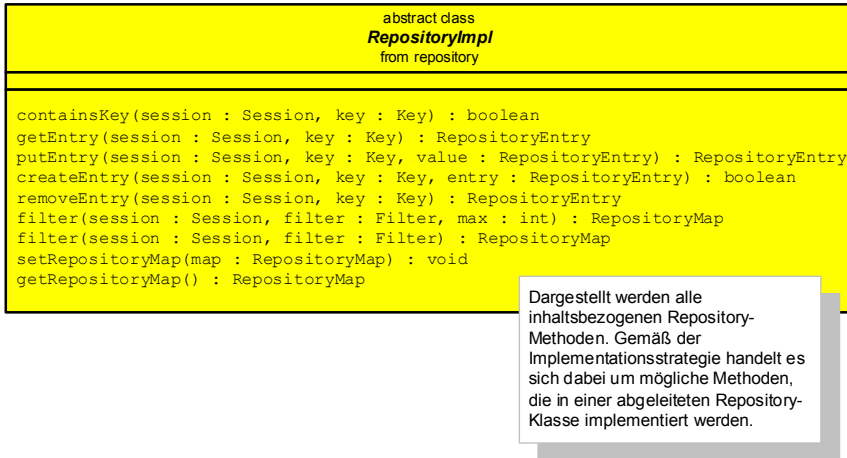


Abb. 39 – Die Klasse `RepositoryImpl` mit allen für Datenquellen relevanten Methoden

Die von der Klasse `RepositoryImpl` abgeleitete Klasse hat einen eigenen Persistenz-Mechanismus. Sie leitet die Persistenz-Aufgabe z. B. an ein LDAP- oder DBMS-System (etwa über eine JDBC-Verbindung auf ein RDBMS).

Für eine `RepositoryImpl`-fähige Implementation müssen alle spezifizierten Methoden implementiert werden. Für alle `Repository`-Klassen, die nur bestimmte Methoden unterstützen, sollten die nicht unterstützten Methoden eine sinnvolle Exception werfen.

Ableiten von der Klasse `ApplicationImpl`

Die Klasse `ApplicationImpl` und die von ihr abgeleiteten Klassen haben folgende Aufgaben:

- Verwalten aller Repositories, die mit einer `Application` assoziiert sind
- Bereitstellen der Business-Logik für eine `Application`

Alle Methoden, die für die Unterstützung des Portal Manager API benötigt werden, werden durch die Basisklasse `ApplicationImpl` zur Verfügung gestellt und müssen nicht durch eine abgeleitete `Application`-Klasse implementiert werden. Die Konfiguration einer `Application` wird über den Admin-Client vorgenommen. Allgemeine Richtlinien zum Implementieren der Business-Logik für eine bestimmte `Application` gibt es nicht.

Ableiten von der Klasse `SessionApplication`

Die Klasse `SessionApplication` (und die von ihr abgeleiteten Klassen) haben folgende Aufgabe:

- Kapseln einer `Application` als `JavaBean` zur Unterstützung der JSP-`Application`
- Assoziieren einer `Application` mit einer bestimmten HTTP-Session (und der HTTP-Session mit einem WCM-Benutzer)
- Pflegen eines Runtime-Locale für die `Application`
- Zurückgeben von Meldungen und Ressourcen entsprechend der aktuellen Locale-Einstellung. Dies wird über das Internationalisierungsobjekt, das mit der `SessionApplication` assoziiert ist, vorgenommen.

Alle Methoden, die für die Unterstützung des Portal Manager API benötigt werden, werden von der Basisklasse `SessionApplication` zur Verfügung gestellt und müssen nicht von den abgeleiteten Bean-Klassen implementiert werden.

5.4 Gestaltung einer JSP-Seite

Für die Gestaltung einer JavaServer-Page (JSP-Seite) stehen Ihnen verschiedene Modelle zur Verfügung. In der JSP-Modell-1-Architektur ist die JSP-Seite dafür verantwortlich, eine Anfrage zu bearbeiten und eine Antwort an den Client zu schicken. Der Zugriff auf die Daten wird von den JavaBeans durchgeführt. Dieses Modell führt zu einer hohen Anzahl von Scriptlets und/oder Java-Code in der JSP-Seite. Dadurch wird es erschwert, diese Quellen zu pflegen, Fehler in ihnen zu beseitigen oder sie erneut zu verwenden. Außerdem gibt es keine klare Trennung zwischen der Logik und der Präsentation. Dadurch ist es schwierig, die Aufgaben zwischen Java-Entwickler und Webdesigner aufzuteilen.

Das JSP-Modell-2 verwendet das MVC-Entwurfsmuster (Model View Controller). Hier wird eine *Controller*-JSP-Seite oder ein Servlet eingesetzt, die für das Bearbeiten der Anfrage und das Erzeugen der Beans verantwortlich sind. Die JavaBeans (Portal Manager API-Klassen) sind das *Model*, das Zugriff auf die Datenquellen hat. Der Controller entscheidet – in Abhängigkeit von den Anfrageparametern –, an welche JSP-Seite die Anfrage weitergeleitet wird. Die *View*-JSP-Seite hat keine Verarbeitungslogik. Sie verwendet die Beans zum Aufbau des dynamischen Contents.

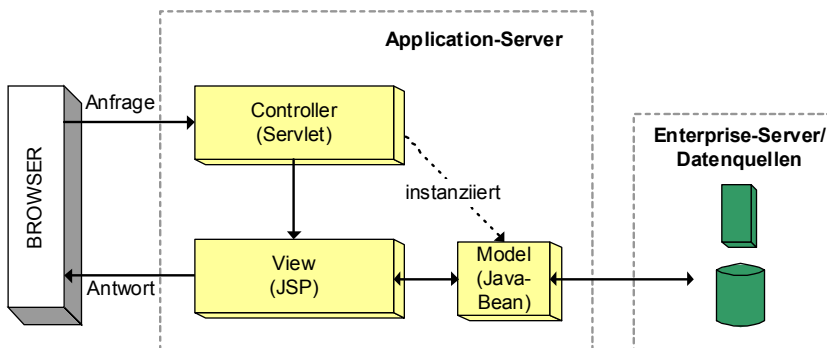


Abb. 40 – JSP-Modell-2-Architektur

Tipps zum Entwickeln von JSP-Seiten mit dem Portal Manager API

Livelihood WCM Server bietet zwei Möglichkeiten, um den Controller der JSP-Modell-2-Architektur zu implementieren. Zunächst einmal können Sie die JSP-Seite oder das Servlet als Controller verwenden. Da es mit Livelihood WCM Server einfacher ist, JSP-Seiten als Servlets zu handhaben, wird der Einsatz von JSP-Seiten empfohlen. Darüber hinaus können Sie auch mit WCM-Vorlagen arbeiten, die folgende Funktionalität beinhalten können:

- Anmeldungsüberprüfung, ob der Benutzer der Session sich bereits am Portal Manager API angemeldet hat
- Erstellen und Initialisieren der Beans des Portal Manager API, die in der Ansicht benötigt werden
- Weiterleiten der Anfrage an eine JSP-Seite, je nach Anfrageparameter
- Fehlerbehandlung mithilfe eines Try-Catch-Blocks

```
<%
    try {
%>
{VIPCONTENT}
<%
        } catch (Exception ex) {
            //Fehlerbehandlung
%>
Fehler: <%=ex.getMessage()%>
<%
    }
%>
```

Obwohl die erzeugte Seite die Vorlage und den Content der View (wie im JSP-Modell-1) beinhaltet, kann der Webdesigner den Inhalt der Vorlage nicht sehen. Deshalb sollten die Vorlagen von erfahrenen Java-Entwicklern geschrieben werden.

Der Webdesigner ist für das Schreiben der View (des WCM-Objekts) verantwortlich. Wenn er ein WCM-Objekt erstellt, kann er eine Vorlage auswählen, die sich später gegebenenfalls auch noch ändern lässt. Der Webdesigner sollte in der JSP-Seite so wenig Java wie möglich verwenden.

Wenn der Webdesigner den Einsatz von Java nicht vermeiden kann, z. B. aufgrund von logischen oder arithmetischen Operationen, die nicht das Design betreffen, sollte er seinen Code in JavaBeans bzw. -Klassen auslagern.

5.5 Sicherheitsaspekte

Für das Design einer Webanwendung, die mit sensiblen Daten operiert, ist der Sicherheitsaspekt besonders wichtig. Neben Aspekten wie Authentifizierung, Verschlüsselung etc., die in die Anwendung implementiert werden müssen, ist es auch wichtig, sich den Zugriff für die Java Virtual Machine (JVM) bewusst zu machen, die die serverseitige Software ausführt. Im Fall der Verwendung von JSP-Seiten muss ein genauerer Blick auf die Sicherheitsmechanismen der JVM geworfen werden, die innerhalb der JSP-Engine verwendet wird.

JVM ohne SecurityManager

Wenn im Kontext einer Anwendung kein SecurityManager explizit gesetzt ist, verfügt die JVM über die Zugriffsrechte des Benutzers, der die Anwendung gestartet hat. Abhängig von der Funktionalität der Anwendung kann dies zu gravierenden Sicherheitslücken führen. Mögliche Szenarien:

- Die Anwendung und damit auch der Client können im serverseitigen Dateisystem navigieren und somit Content ansehen bzw. ändern.

- Die Anwendung und somit auch der Client können eventuell Netzwerkverbindungen zu Computern herstellen, die in der serverseitigen Umgebung liegen.

In einer solchen Umgebung wird empfohlen, dass dem oben erwähnten Benutzer nur die Rechte zugestanden werden, die dafür sorgen, dass die Anwendung fehlerfrei läuft. Außerdem sollte der Zugriff auf andere Ressourcen, besonders auf sicherheitsrelevante oder sensible Daten, verboten sein.

JVM mit SecurityManager

Durch die Java-2-Plattform steht eine überarbeitete Sicherheitsarchitektur mit fein strukturierten und leicht konfigurierbaren Sicherheitsmechanismen zur Verfügung. Um diese Funktionen nutzen zu können, führen Sie die folgenden Schritte aus:

1. Integrieren Sie einen SecurityManager in den Serverprozess.
2. Erzeugen Sie eine Sicherheitsmechanismus-Datei, die der Anwendung die Rechte bereitstellt, die für eine fehlerlose Operation notwendig sind. Diese Datei kann verwendet werden, um den Zugriff einer Anwendung entscheidend einzuschränken, verglichen mit den Rechten, die einem Benutzer vom Betriebssystem gegeben werden.

Eine detaillierte Beschreibung des SecurityManager finden Sie in der Dokumentation zu Java 2.

5.6 Internationalisierung

Das Internet und Intranet-Anwendungen werden zunehmend global verwendet und für Benutzer in verschiedenen Ländern ausgelegt.

Dadurch besteht ein potenzieller Bedarf, eine auf dem Portal Manager API basierende Anwendung zu internationalisieren.

Die Applications basieren auf Java, sodass sie automatisch die Internationalisierungsunterstützung erben, die Java bietet. Dabei handelt es sich um:

- `Locale` – das `Locale`-Objekt als einfacher Identifier für eine Sprache (oder eine Region)
- lokalisierte Ressourcen – das `ResourceBundle` als Container für alle `Locale`-spezifischen Befehle und Objekte
- Kalender- und Zeitzonenunterstützung
- `Locale`-spezifische Formatierung von Zahlen, Daten etc.
- `Locale`-spezifische String-Methoden (z. B. `Locale`-spezifisches Sortieren)

Sämtliche Funktionen sind detailliert in der Spezifikation von Java 2 beschrieben.

Als Webanwendung verwendet das Portal Manager API neben der Programmiersprache Java weitere Softwarekomponenten. Die Architektur und die Implikation dieser Architektur für die Internationalisierung werden in den folgenden Abschnitten erklärt.

Architektur des Portal Manager API und Internationalisierung

Im folgenden Diagramm wird die Architektur des Portal Manager API mit allen für die Internationalisierung relevanten Aspekten dargestellt.

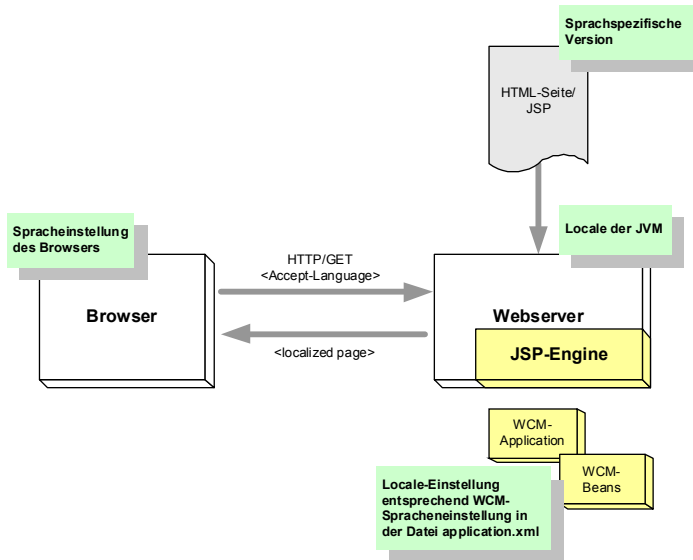


Abb. 41 – Architektur des Portal Manager API mit Aspekten der Internationalisierung

Die Architektur basiert auf den folgenden Komponenten:

- ein Browser als Client (für das HTTP-Protokoll)
- ein Webserver als Server (für das HTTP-Protokoll)
- eine JSP-Engine als Adapter zwischen Webserver und Java-Anwendungen
- Komponenten des Portal Manager API (z. B. JavaBeans), die in einer JSP-Engine ausgeführt werden

Wie im Diagramm beschrieben, verfügt jede dieser Komponenten über eigene sprach- bzw. internationalisierungsrelevante Einstellungen:

- Der Browser kann eine bestimmte Sprache über den HTTP-Header anfordern.
- Der Webserver muss verschiedene Dokumentversionen (HTML, JSP-Seite, Grafiken usw.) bereithalten.
- Die JVM, die die JSP-Engine ausführt, hat eine standardmäßige Locale-Einstellung.
- Die WCM-Beans und WCM-Application sind Locale-spezifisch. Die standardmäßige Spracheinstellung für eine Application und das betreffende Bean wird im Admin-Client festgelegt (siehe Abschnitt 3.2 “Applications verwalten” ab Seite 69).

Hinweis: Das standardmäßige Locale für das SessionBean wird über die Standardsprache des angemeldeten WCM-Benutzers festgelegt.

Vorschlag für die Internationalisierung einer Portal Manager API-Anwendung

Basierend auf der Architektur, die in Abschnitt “Architektur des Portal Manager API und Internationalisierung” auf Seite 171 beschrieben wurde, könnte eine Internationalisierung nach der folgenden Strategie durchgeführt werden: Der Client, der die Anwendung verwendet, ist der Empfänger aller Informationen bzw. des gesamten Contents. Daher gibt er seine bevorzugte Sprache bzw. Lokalisierung an. Alle anderen Systemkomponenten müssen ihr Verhalten entsprechend der Auswahl des Clients anpassen.

Diese Strategie führt zu den folgenden technischen Empfehlungen:

- Der Benutzer kann seine bevorzugte Sprache im Browser festlegen. Diese Präferenz wird im Header jeder HTTP-Anfrage gesendet und kann vom Webserver bzw. von der JSP-Engine ermittelt werden. Die Sprachauswahl gemäß RFC 1766 und ISO 639 kann dem Servlet-Request entnommen werden.
- Die URL kann dynamisch erstellt werden und auf den entsprechenden Content verweisen.
- Für die Internationalisierung von dynamischem Content, der von Applications verfügbar gemacht wird, muss Folgendes gelten:
 - Die JavaBeans bzw. Applications müssen die Internationalisierung unterstützen.
 - Eine JSP-Seite, die ein bestimmtes Bean oder eine bestimmte Komponente instanziiert, muss das `Locale` entsprechend den Benutzereinstellungen festlegen. Dies geschieht, indem der Wert `accept-language` aus dem Header des `HttpRequest`-Objekt abgefragt wird und ein entsprechendes Java-`Locale`-Objekt erzeugt wird.

Das folgende Code-Beispiel zeigt eine JSP-Seite, die die Internationalisierung mit einem Session-Bean verwendet.

```
...  
  
<body>  
  
<%@ page import="java.util.Locale" %>  
<h1>Setting the Locale for a WCM bean from the HTTP header</h1>  
<jsp:useBean id="sessionBean"  
    class="de.gauss.vip.portalmanager.SessionBean" />  
<%  
    sessionBean.init(request);  
    Locale locale = request.getLocale();  
    sessionBean.setLocale(locale);  
    out.println("sessionBean locale=" + sessionBean.getLocale()  
        + "<br>");
```

```
        out.println(sessionBean.getI18n().getString("MY_MESSAGE_KEY"));
    %>

</body>

...

```

Unterstützung mehrerer Sprachen durch ein zentrales Template

Die Struktur einer Website könnte sich wie in folgender Abbildung darstellen:

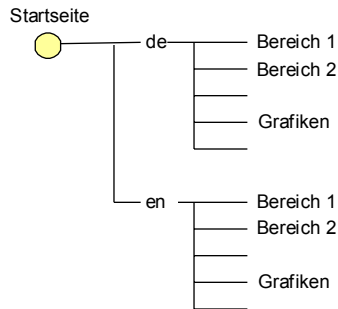


Abb. 42 – Struktur einer Website

Wenn der Benutzer auf die Startseite der Website gelangt, kann er eine Sprache auswählen, z. B. Englisch. Nach Auswahl der Sprache wird eine JSP-Seite aufgerufen, die alle erforderlichen OIDs in der Session setzt wie z. B. die der Startdateien für die einzelnen Bereiche. Wenn ein Bereich ausgewählt wird, erhält das Template die OID aus der Session und zeigt die entsprechenden Objekte an.

Applications und Internationalisierung

Das Portal Manager API bietet die Möglichkeit, benutzerspezifische Applications in das API zu integrieren. Damit Sie die Internationalisierungsunterstützung des Portal Manager API nutzen können, führen Sie die folgenden Schritte durch.

1. Lokalisieren Sie alle Ressourcen entsprechend den Standards, die in Java 2 (PropertyResourceBundle usw.) festgelegt sind.
2. Ordnen Sie der benutzerspezifischen Application die Ressourcen und Parameter zu (siehe dazu den Abschnitt 3.2 "Applications verwalten" ab Seite 69).

WCM-Beans und Internationalisierung

Alle WCM-Beans verfügen über eine integrierte Unterstützung für die Internationalisierung. Die Verwendung wird durch die folgenden Methoden ermöglicht:

- Die aktuelle Einstellung des Locale, das für ein bestimmtes Bean gültig ist, wird durch `public java.util.Locale getLocale` zur Verfügung gestellt.
- Das Locale für ein bestimmtes Bean kann gesetzt werden durch `public void setLocale(java.util.Locale locale)`.

WCM-Internationalisierungsobjekt

Um die Internationalisierung zu unterstützen, stellt Livelink WCM Server ein spezifisches Internationalisierungsobjekt zur Verfügung, das in den WCM-Beans verwendet wird. Der Zugriff auf dieses Objekt wird über die Methode `getLocale` gewährleistet. Dieses Objekt kann auch für die Internationalisierung von eigenen Beans und Applications verwendet werden.

Die Klasse `de.gauss.vip.i18n.Internationalization` stellt folgende Funktionalität zur Verfügung:

- Laden von `ResourceBundle`-Objekten
- Erhalten von Schlüsseln von einem bestimmten `ResourceBundle`
- Erhalten von `Locale`-spezifischen Daten- und Zeitformaten

ANHANG A

Metadatenschemata

Um die Abwärtskompatibilität mit den alten Versionen des Portal Manager API zu gewährleisten, sind die Namen und Datentypen der Metadaten gegenüber der Version 5e unverändert geblieben.

Bei der Konfiguration jedes `VipObjectRepository` und `VipObjectHandlerRepository` können Sie angeben, welches Metadatenchema Sie verwenden möchten (siehe Tabelle 3 "Repositories und ihre Parameter" auf Seite 60).

Die folgende Tabelle stellt die Metadatenchemata für die Versionen 5e und 8 gegenüber.

Metadatenchema in Version 5e		Metadatenchema in Version 8	
Attribut	Wert	Attribut	Wert
acl	VipAclValue	acl	Acl
author	StringValue	created_by	User
date-created	DateValue	date_created	DateValue
date-expire	DateValue	date_expire	DateValue
date-release	DateValue	date_released_at	DateValue
date-released	DateValue	date_released	DateValue
direct-release	StringValue	direct_release	BooleanValue

Metadatenschema in Version 5e		Metadatenschema in Version 8	
Attribut	Wert	Attribut	Wert
edit-url	StringValue	url	StringValue
		Anmerkung: In Repositories vom Typ EDIT	
elements	RepositoryMap	all_children	ListValue
email-edit	StringValue	email_edit	SetValue
email-qa	StringValue	email_qa	SetValue
email-release	StringValue	email_release	SetValue
filename	StringValue	filename	StringValue
		Anmerkung: Portal Manager API-spezifisch	
generic1	StringValue	keywords	SetValue
generic2	StringValue	description	StringValue
generic3	StringValue	target_group	StringValue
language	StringValue	language	LocaleValue
level	IntegerValue	level	IntegerValue
		Anmerkung: Portal Manager API-spezifisch	
linked-from	RepositoryMap	linked_from	SetValue
links-to	RepositoryMap	links_to	SetValue
name	StringValue	title	StringValue
object-icon	StringValue		

Metadatenschema in Version 5e		Metadatenschema in Version 8	
Attribut	Wert	Attribut	Wert
oid	StringValue	oid	ObjectId
pathname	StringValue	pathname	StringValue
prod-url	StringValue	url	StringValue
		Anmerkung: In Repositories vom Typ PROD	
qs-url	StringValue	url	StringValue
		Anmerkung: In Repositories vom Typ QA	
status	StringValue	state	ObjectState
status-icon	StringValue		
		Anmerkung: erreichbar über ObjectState	
subtitle	StringValue	subtitle	StringValue
suffix	StringValue	suffix	StringValue
		Anmerkung: Portal Manager API-spezifisch	
template	StringValue	template	ObjectId
topic	StringValue	topic	ObjectId
topics	RepositoryMap		
type	StringValue	type	ObjectType
url	StringValue	url	StringValue

Metadatenschema in Version 5e		Metadatenschema in Version 8	
Attribut	Wert	Attribut	Wert
version	IntegerValue	version	Version
vip-edit-url	StringValue	surrogate_url	StringValue
		Anmerkung: In Repositories vom Typ EDIT	
vip-qs-url	StringValue	surrogate_url	StringValue
		Anmerkung: In Repositories vom Typ QA	
website	StringValue	website	StringValue
@@codebase	StringValue		
		category	StringValue
		date_modified	DateValue
		deployment_hint	StringValue
		Anmerkung: Enthält Dateinamen mit Endung	
		modified_by	User
		released_by	User

ANHANG B

Caching

Für den Betrieb von Livelink WCM Server ist ein relationales Datenbank-Managementsystem erforderlich. In der Datenbank werden u. a. die Metadaten der WCM-Objekte gespeichert. Dieser Anhang beschreibt das Caching-Verhalten des Portal Manager API in Bezug auf Metadaten.

B.1 Zugriff auf WCM-Objekte

Beim ersten Zugriff auf ein Objekt werden die gespeicherten Metadaten aus der Datenbank geladen. Das Objekt wird aus den Metadaten erstellt und anschließend im serverseitigen Cache zwischengespeichert. Wenn ein bereits geladenes Objekt erneut angefordert wird, wird das im Cache enthaltene Objekt zurückgeliefert.

Sobald sich ein Objekt durch eine Staging-Aktion ändert, wird es aus dem Cache entfernt. Dadurch müssen beim nächsten Zugriff auf das Objekt die aktualisierten Metadaten aus der Datenbank neu geladen werden. Da ein Objekt in jeder Sicht (Edit, QS und Produktion) unterschiedliche Metadaten enthalten kann, wird für jede Sicht ein eigenes Objekt in der Datenbank und im Cache gehalten.

Die Erstellung einiger Metadaten ist sehr zeitaufwendig. Da der Content-Client in der Regel nicht alle Metadaten benötigt, werden beim Laden eines Objekts nur *Basisdaten* aus der Datenbank gelesen. Dadurch wird im Server weniger Speicher verbraucht und insgesamt der Ladevorgang beschleunigt. Die nicht geladenen Metadaten – die so genannten *verzögert zu ladenden Daten* – werden bei Bedarf aus der Datenbank gelesen.

Sämtliche Spezialattribute sowie die folgenden Daten werden in Livelink WCM Server verzögert geladen:

- pathname
- filename
- suffix
- url
- surrogate_url
- links_to
- linked_from
- all_children (nur für Oracle)

Hinweise

- Wenn Sie das Nachladen aus der Datenbank während des laufenden Betriebs vermeiden möchten, sollten Sie direkt nach dem Laden des Objekts auf die verzögert zu ladenden Daten zugreifen. Dadurch können diese Daten zu einem späteren Zeitpunkt aus dem Cache geliefert werden.
- Die Metadaten `links_to` und `linked_from` sollten nur in Ausnahmefällen geladen werden, da sie nur durch eine zeitaufwendige Datenbankabfrage ermittelt werden können.

- Für das Metadatum `linked_from` gilt, dass das Objekt nicht aus dem Cache entfernt wird, wenn einem anderen Objekt eine Referenz auf das Objekt hinzugefügt wird oder eine Referenz auf das Objekt entfernt wird.

B.2 Deployment-Cache

Deploymentsysteme haben in Livelink WCM Server einen eingebauten Cache für die vom Deployment gelieferten Metadaten, sodass diese Daten ab dem zweiten Zugriff direkt geliefert werden können. Voraussetzung hierfür ist, dass das Deploymentsystem auf dem zugreifenden Server (i.A. ein Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft) eingerichtet ist. Diese Konstellation ist zu empfehlen, wenn im normalen Betrieb nur wenige Änderungen durchgeführt werden, die große Auswirkungen für das Deployment haben. Andernfalls kann der für das Deployment erforderliche zusätzliche Aufwand auf diesem Content-Server zu Engpässen führen.

Es ist im Einzelfall abzuwägen, ob ein schnellerer Zugriff auf die Daten des Deploymentsystem die zusätzliche Belastung durch das separate Deploymentsystem aufwiegt. Beispiel: Sie verwenden in Ihrer Website eine Vorlagenkaskade, die mehr als 10.000 Objekten zugeordnet ist. Einige der Vorlagen werden regelmäßig einmal pro Woche geändert. Nach der Freigabe der geänderten Vorlagen hat das zur Folge, dass alle betroffenen Objekte neu erzeugt werden müssen. Dies führt zu einer erhöhten Belastung des Servers. Ob für Ihr System eine erhöhte Belastung akzeptabel ist, kann durch Performance-Tests ermittelt werden.

B.3 Funktionsweise des Caches für WCM-Objekte

Der Cache für WCM-Objekte wird im Admin-Client in den Einstellungen der Website konfiguriert (*Konfiguration* → *Websites* → *{Website-Name}* → Registerkarte *Allgemein* → *Caching*). Dort lassen sich folgende Parameter einstellen:

- *Minimale Größe*: Anzahl von Objekten, auf die der Cache in Leerlaufzeiten des Servers reduziert wird. Der Standardwert ist 20000.
- *Maximale Größe*: Maximale Anzahl von Objekten im Cache. Der Standardwert ist 45000.
- *Cache-Reduzierzeit*: Zeitspanne in Millisekunden. Der Standardwert ist 7200000 (entspricht 2 Stunden). Wird während dieses Zeitraums nicht auf die Objekte im Cache zugegriffen, werden Objekte aus dem Cache entfernt, bis wieder die minimale Größe erreicht ist.

Wenn auf ein Objekt zugegriffen wird, wird zuerst im Cache nach dem Objekt gesucht. Ist es dort nicht vorhanden, wird es aus der Datenbank geladen. Anschließend wird das Objekt im Cache abgelegt. Hierbei werden drei Fälle unterschieden:

- **Die minimale Größe wurde noch nicht erreicht.**

Das neu geladene Objekt wird an einer freien Position im Cache abgelegt. Der Cache vergrößert sich.

- **Die minimale Größe wurde erreicht, die maximale Größe noch nicht.**

Das neu geladene WCM-Objekt wird im Cache abgelegt. Es ist möglich, dass dabei ein Objekt aus dem Cache entfernt wird, sodass sich die Größe des Caches nicht unbedingt erhöht.

- **Die maximale Größe wurde erreicht.**

Das neu geladene WCM-Objekt wird im Cache abgelegt. Der Cache ermittelt auf Basis eines bestimmten Algorithmus (Häufigkeit der Zugriffe/Verweildauer im Cache), welches Objekt aus dem Cache entfernt wird.

Ist bei Erreichen der eingestellten Cache-Reduzierzeit der Cache größer als im Parameter für die minimale Größe angegeben, entfernt der Cache selbstständig so viele Objekte, bis wieder die minimale Größe erreicht ist. Dieser Vorgang wird im eingestellten Zeitabstand wiederholt.

Wenn Sie sicherstellen möchten, dass viele Objekte im Cache gehalten werden, setzen Sie den Wert für die minimale Größe entsprechend hoch. Sie verhindern so ein frühzeitiges Entfernen der Objekte aus dem Cache. Die maximale Größe muss entsprechend angepasst werden. Insbesondere auf Content-Servern mit Produktionssicht sollte versucht werden, einen Großteil der Objekte während des Betriebs im Cache zu halten.

B.4 Cachen von RepositoryEntry-Objekten

Die Darstellung der Metadaten am Content-Server, der im Kontext einer JSP-Engine bzw. als Webanwendung in einem Application-Server läuft, die sogenannte *externe* Repräsentation, wird durch RepositoryEntry-Objekte realisiert. Die externe weicht teilweise von der internen Darstellung der Daten ab. Um den Konvertierungsaufwand so gering wie möglich zu halten, hat jedes im Cache befindliche Objekt einen eigenen Cache für die Speicherung der RepositoryEntry-Objekte. Sobald ein Objekt aus dem Cache entfernt wird, werden auch die entsprechenden RepositoryEntry-Objekte entfernt.

Wenn der Content-Server ein `RepositoryEntry` anfordert, prüft das WCM-Objekt, ob die gewünschte Attributkombination in seinem Cache bereits vorhanden ist und liefert die Daten direkt von dort zurück. Andernfalls werden die gewünschten Daten entsprechend aufbereitet, wobei sie dazu gegebenenfalls aus der Datenbank nachgeladen werden müssen. Die Attributkombination wird anschließend mit ihren Werten im Cache abgelegt.

Hieraus ergibt sich, dass Sie beim Programmieren von JSP-Seiten möglichst alle auf der Seite benötigten Attribute beim ersten Zugriff des `RepositoryEntry` mit angeben sollten. Dadurch vermeiden Sie, dass bei abweichenden Attributen erneut auf die Datenbank zugegriffen wird und die Datenbank Werte zurückliefern muss.

Glossar

Application – Schnittstelle zur einheitlichen Beschreibung einer externen Anwendung

Context-ID – Objekt, das einem Benutzer nach erfolgreicher Anmeldung am WCM-System zugeteilt wird. Eine Context-ID ist immer systemweit eindeutig, sie identifiziert deshalb genau einen Benutzer. Wird eine Context-ID eine gewisse Zeit nicht gebraucht, so verfällt sie.

Deploymentsystem – Die Deploymentsysteme erzeugen aus den WCM-Objekten Seiten und verteilen die generierten Dateien in die dafür vorgesehenen Verzeichnisse. Von dort aus werden die Dateien über den Einsatz eines HTTP-Servers für die Benutzer sichtbar. Deploymentsysteme können unterschiedliche Typen und Kategorien haben.

Dynamisierung – Der Inhalt der einzelnen Seiten einer Website wird komplett oder teilweise erst zur Laufzeit erzeugt.

JSP-Engine – Im Webserver integriertes Modul zur Ausführung von JSP-Skripts, die in HTML-Seiten eingebettet sind. JSP-Engines enthalten im Allgemeinen Java-Compiler.

JSP-Skript – HTML-Seite, in die Java-Code eingebettet wurde, der serverseitig ausgeführt wird

LoginRepository – Repository, mit dem Informationen zu Benutzer-Accounts verarbeitet werden

Master-Server – Nur Master-Server haben lesenden und schreibenden Zugriff auf die Daten eines WCM-Systems. Der Master-Content-Server verwaltet die Website-Daten, der Master-Administrationsserver die Konfigurations- und Systemdaten des WCM-Systems. Siehe auch *Serverkategorie*.

ObjectHandler – Diese Instanz nimmt Aktionen an und manipuliert so die Objekte der Website.

Personalisierung – Dynamisierung der Website, die auf den aktuellen Besucher individuell zugeschnitten ist

Portal – Als Portal wird eine Website bezeichnet, die dem Benutzer als zentraler Einstieg – als Tor – in bestimmte Internetangebote dient. Ein Portal bietet oftmals themenbezogene, personalisierte Angebote und Informationen.

Properties-Datei – Datei mit Ressourcen-Informationen, die in einem definierten Format (Schlüssel-Wert-Paare) vorliegen

Proxy-Server – Ein Proxy-Server dient dazu, Anforderungen von einer Client-Anwendung, etwa einem Browser, an einen anderen (oder mehrere andere) Server abzufangen. Wenn der Proxy-Server die Anforderung erfüllen kann, sendet er die angeforderten Daten an die Client-Anwendung zurück. Andernfalls leitet er die Anforderung an den spezifizierten Server weiter.

Im Kontext von Livelink WCM Server haben WCM-Server der Kategorie "Proxy" keinen schreibenden, sondern nur lesenden Zugriff auf die WCM-Objekte bzw. die Konfiguration. Die Änderung von WCM-Objekten erfolgt ausschließlich über den Master-Content-Server, Änderungen an der Konfiguration des WCM-Systems sind nur über den Master-Administrationsserver möglich. Siehe auch *Serverkategorie*.

Repository – Repository ist eine Abstraktion, die einen einheitlichen Zugriff auf Datenbestände erlaubt.

Serverkategorie – In einem WCM-System wird zwischen Master- und Proxy-Servern unterschieden. Master-Server haben schreibenden Zugriff auf die Daten des WCM-Systems, Proxy-Server dagegen nur lesenden. Der Master-Content-Server verwaltet die Website-Daten, der Master-Administrationsserver die Konfigurations- und Systemdaten. Darüber hinaus können beliebig viele Proxy-Server eingesetzt werden.

Servertyp – Entsprechend den Aufgaben der Server gibt es zwei Servertypen: Content-Server zur Verwaltung von Website-Daten und Administrationsserver zur Verwaltung der Benutzer-, Konfigurations- und Systemdaten des WCM-Systems. Jeder Content-Server ist grundsätzlich dazu in der Lage, alle Sichten auf die Daten der verwalteten Website(s) zur Verfügung zu stellen – Edit, QS und Produktion. Die verfügbaren Sichten können dadurch eingeschränkt werden, dass der Content-Server nur die Daten von bestimmten Sichten erhält.

Session – Einheit, die von der JSP-Engine verwaltet wird, damit logisch zusammenhängende Aktionen (aus Sicht der Ressourcen) auch zusammengefasst werden können

Session-Bean – JavaBean, das in einem JSP-Skript in Verbindung mit dem Portal Manager API benutzt werden kann. Die Lebensdauer eines Session-Beans erstreckt sich über die gesamte Session.

Statifizierung – Beim Statifizieren werden die dynamischen Bestandteile z. B. einer JSP-Seite in statische umgewandelt. Das Ergebnis ist reines HTML ohne Java-Code.

–

Index

A

- absolute_urls (Spezialattribut) 41
- Anlegen
 - Application 70
 - Repository 56
- Anwendungsbeispiele
 - Anlegen von Benutzern in LDAP über ein Portal 140
 - Dynamische Navigationselemente 150
 - HTML-Formular 158
 - Multicontent-Seite 154
- Anwendungsintegration 36
- Application (Interface) 117
- ApplicationImpl (Klasse) 118
- Applications 116
 - anlegen 70
 - Einleitung 22
 - Framework 160
 - implementieren 162
 - Internationalisierung 175
 - Klassenname 71
 - logischer Name 72
 - löschen 88
 - Parameter der mitgelieferten Applications 76
 - Parameter festlegen 75
 - Repository zuordnen 86
 - Repository zuordnen (beim Anlegen) 73
 - Ressourcen 74
 - Sprache 72
 - verwalten 69
 - zu Content-Server zuordnen 89
- Architektur
 - Portal Manager API 19

Authentifizierung 133

B

- Base (Klasse) 115
- Basisdatentypen 102
- Beans 119
- Benutzer anlegen in LDAP über Portal 140

C

- Content
 - dynamischer 33
- ContentHandler (Interface) 131
- Content-Objekte
 - platzieren mehrerer 36
- Content-Server in Application-Server starten 31

D

- Datenwerte (Values) 103
- DirectoryRepository
 - Klasse 91
- Dynamische Navigationselemente 150
- Dynamischer Content 33

F

- Fehlersuche 94
- Filterung 105
- FormData (Klasse) 37
- Formulare
 - Anlegen eines Objekts 38
 - Umgang mit 37
- Framework
 - Applications 160

G

- generate_static (Spezialattribut) 40
- Grundlegende Klassen 102
 - Basisdatentypen 102
 - Filterung und Sortierung 105
 - Repositories 111
 - RepositoryMap 107

H

- HTML-Formular 158

I

- Implementieren
 - Applications 162
- Integration
 - externe Anwendungen 48
 - in das WCM-System 24
- Interfaces
 - Application 117
 - ContentHandler 131
 - RepositoryEntryIterator 109
 - RepositoryIterator 110
 - Value 103
 - VipProfile 123
- Internationalisierung 170
 - Architektur des Portal Manager API 171
 - Portal Manager API-Anwendung 172
 - WCM-Internationalisierungsobjekt 175
- Internationalization (Klasse) 122

J

- Javadoc 101
- Java-Server-Page
 - siehe JSP-Seite
- JSP-Seite
 - Gestaltung 166
 - Modell-1-Architektur 166

- Modell-2-Architektur 166

- JSP-Skript 20

- JVM

- mit SecurityManager 169
- ohne SecurityManager 168

K

- Klassen

- ApplicationImpl 118
- Base 115
- FormData 37
- Internationalization 122
- RepositoryEntry 108
- RepositoryImpl 113
- RepositoryMap 109
- SessionApplication 121
- SessionBean 123
- VipObjectBean 126
- VipObjectFilterBean 130
- VipObjectHandlerBean 46, 128
- VipUserBean 126
- VipWhatsNewBean 130

- Klassenbeschreibung 101

- Klassendiagramme 99

L

- LDAP

- DirectoryRepository (Klasse) 91

- Lebenszyklus eines WCM-Objekts 46

- Locale 170

- Login-Repositories 112

- Löschen

- Application 88
- Repository 68

M

- Metadatenschemata 179

- Multicontent-Seite 154

N

Navigationselemente 34

Neu

 Application 70

 Repository 56

Neue Benutzer in LDAP anlegen 140

P

Parameter

 einer Application festlegen 75

 eines Repository festlegen 58

 mitgelieferte Applications 76

 VipDAVApplication 77

 VipHCLApplication 78

 VipHTMLClientApplication 80

 VipObjectApplication 84

 VipUserApplication 84

 WebServiceApplication 84

Personalisierung 35

Platzierung mehrerer Content-Objekte
36

Portal

 Benutzer in LDAP anlegen 140

Portal Manager API

 Anwendungsbeispiele 140

 Architektur 19

 Integration 24

 Staging 46

Profil 123

R

Repositories 111

 anlegen 56

 Application zuordnen 65

 Einleitung 23

 Klassenname 57

 Login-Repositories 112

 löschen 68

 Name 57

 Parameter festlegen 58

 Typen 112

 verwalten 55

RepositoryEntry (Klasse) 108

RepositoryEntryIterator (Interface) 109

RepositoryImpl (Klasse) 113

RepositoryIterator (Interface) 110

RepositoryMap 107

RepositoryMap (Klasse) 109

S

Schlüsselwerte (Keys) 103

SecurityManager

 JVM mit 169

 JVM ohne 168

Session 21, 113

SessionApplication (Klasse) 121

SessionBean (Klasse) 123

Session-Beans 21

Session-Management 133, 137

Sicherheitsaspekte 168

Sortierung 106

Spezialattribute

 Statifizierung 39

Sprache

 Application 72

Staging 46

Starten

 Content-Server in Application-
 Server 31

Statifizierung 39

 absolute_urls (Spezialattribut) 41

 generate_static (Spezialattribut) 40

 timeout (Spezialattribut) 40

 von Formularinstanzen 43

 von JSP-Objekten 42

 von XML-Objekten 45

Systemarchitektur

 Master-Administrationsserver 26

- Master-Content-Server 26
- Minimalsystem 28
- Proxy-Server 27
- separate Datenhaltung 29

T

- timeout (Spezialattribut) 40
- trace.properties 94

V

- Value (Interface) 103
- Verwalten
 - Applications 69
 - Repositories 55
- VipDAVApplication
 - Parameter 77
- VipHCLApplication
 - Parameter 78
- VipHTMLClientApplication
 - Parameter 80
- VipObjectApplication
 - Parameter 84
- VipObjectBean (Klasse) 126
- VipObjectFilterBean (Klasse) 130
- VipObjectHandlerBean (Klasse) 46, 128
- VipProfile (Interface) 123
- VipUserApplication
 - Parameter 84
- VipUserBean (Klasse) 126
- VipWhatsNewBean (Klasse) 130

W

- WCM-Objekt
 - Lebenszyklus 46
- WebServiceApplication
 - Parameter 84